MANIPAL INSTITUTE OF TECHNOLOGY
BENGALURU
(A constituent unit of MAHE, Manipal)

B.TECH. FIFTH SEMESTER

Information Technology
(B. Tech CSE (CYBER SECURITY))

COMPUTER NETWORKS LAB

CSE_3162

LABORATORY MANUAL

CONTENTS

BASIC SKILL SETS NEEDED

Basic Computer Networking related commands and configurations

1. Unix Utilities to test Internet connection and to diagnose congestion between computers:

ifconfig: The **ifconfig** command is used to configure a network interface. The following options are used for the reconfiguration of the IP address and network mask.

ifconfig -*a :* Shows the states of all interfaces in the system.

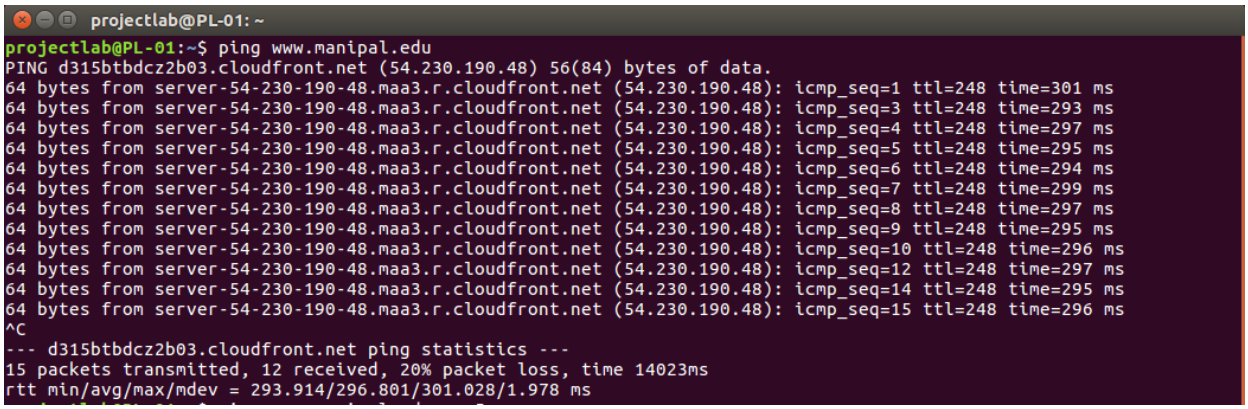
ifconfig -*s :* Display a short list, instead of details.

ifconfig <*interface name*> <*new IP address*> *up* : Assigns a new IP address to the interface and brings it up.

ifconfig<*interface name*>*down*: Disables the network interface, where interface name is the name of the Ethernet interface.

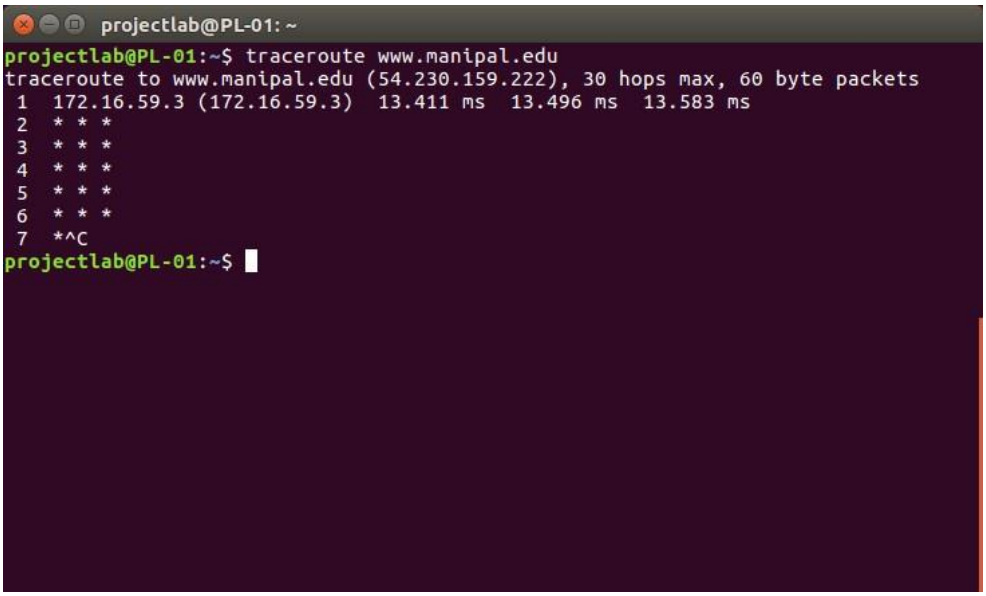
ifconfig <interface name > netmask <new netmask> : Assigns a new network mask for the interface.

Ping: Ping (also written as PING or ping) is a utility that you use to determine whether or not a specific IP address is accessible. Ping works by sending a packet to a specified address and waiting for a reply. Ping is used primarily to troubleshoot Internet connections and there are many freeware and shareware Ping utilities available for download.



Traceroute: Traceroute is a utility that traces a packet from your computer to an Internet host, but it will show you how many hops the packet requires to reach the host and how long each hop takes. If you're visiting a Web site and pages are appearing slowly, you can use traceroute to figure out where the longest delays are occurring. Traceroute utilities work by sending packets with low time-to-live (TTL) fields. For implementing the traceroute command, you have to first install traceroute. The **TTL** value specifies how many hops the packet isallowed before it is returned. When a packet can't reach its destination because the TTL value is too low, the last host returns the packet and identifies itself. By sending a series of packets and incrementing the TTL value with each successive packet, traceroute finds out who all the intermediary hosts.

```
projectlab@PL-01:~$ sudo apt install traceroute
Reading package lists... Done
Building dependency tree
Reading state information... Done
```



TRY IT OUT:

Connect the computers in local area network

1. Right click on the network manager applet,

    - Go to **Edit connections ->wired tab->add**

2. Put the mac address of the interface you will be configuring. The ifconfig command can show you what the mac address is:

$ ifconfig

eth0 Link encap:Ethernet HWaddr 00:30:1b:b9:53:94
HWaddr 00:30:1b:b9:53:94 = mac address

3. Then click the ipv4 settings tab. set method to manual.

4. click add to add IP address

 # example for computer one would be
address | netmask | gateway
10.0.0.1 | 255.255.255.0 |
 # example for computer two would be
10.0.0.2 | 255.255.255.0 |

5. See if you can ping each other from computer one.

$ ping 10.0.0.2

ping 10.0.0.2 (10.0.0.2) 56(84) bytes of data.

64 bytes from 10.0.0.2: icmp_seq=1 ttl=128 time=0.457 ms

I. Do it Yourself.

Note: Use **man** command to explore various options

1. What is the IP of the machine you are using? Compare it with the IP of your neighbors.
   Are the IPs of your neighbors the same? Why or why not?
2. Use the ping command for the following URLs and record the success or failure statistics along with the average round trip time.
   a) google.com
   b) facebook.com
3. Based on output of **ifconfig -a** command, identify the following
   - Host Id
   - MAC address of your system[physical address]
   - Subnet mask
4. In the LAN, compare your result of Q3, with your neighbor computers. What similarities do you see in the MAC address?
5. With the help **man** pages, comment on the results of **ping** with different options.
6. Ping the computer's loopback IP address. Type the following command:

   ping 127.0.0.1

   The address 127.0.0.1 is reserved for loopback testing. If the ping is successful, then TCP/IP is properly installed and functioning on this computer
7. Assign static IP address/mask with **ifconfig** command to your computer
8. Configure your computer to receive Network Settings from DHCP Server of the Institution.
9. Configure DNS Server address for your host with static IP address.
10. Learn about configuring a host behind proxy server of your Institution
11. Learn about Internet Browser's Network Configurations.

LAB NO: 1 **Date:**

Socket Programming in 'C' using TCP -Iterative Client-Server Programs

Objectives:

- To familiarize yourself with application-level programming with sockets.
- To understand principles of Inter-Process Communication with Unix TCP Sockets.
- To Learn to write Network programs using C programming language.

Prerequisites:

- Knowledge of the C programing language and Linux Networking APIs
- Knowledge of Basic Computer Networking

I. Sockets

Sockets allow communication between two different processes on the same or different machines. To be more precise, it's a way to talk to other computers using standard Unix file descriptors. In Unix, every I/O action is done by writing or reading a file descriptor. A file descriptor is just an integer associated with an open file and it can be a network connection, a text file, a terminal, or something else. To a programmer, a socket looks and behaves much like a low-level file descriptor. This is because commands such as read() and write() work with sockets in the same way they do with files and pipes.

Types of Sockets

There are four types of sockets available to the users. The first two are most commonly used and the last one is rarely used.

- **Stream Sockets:** Delivery in a networked environment is guaranteed. If you send through the stream socket three items "A, B, C", they will arrive in the same order - "A, B, C". These sockets use TCP (Transmission Control Protocol) for data transmission. If delivery is impossible, the sender receives an error indicator. Data records do not have any boundaries.
- **Datagram Sockets:** Delivery in a networked environment is not guaranteed. They're connectionless because you don't need to have an open connection as in Stream Sockets -

you build a packet with the destination information and send it out. They use UDP (User Datagram Protocol).

- **Raw Sockets:** These provide users access to the underlying communication protocols, which support socket abstractions. Raw sockets are not intended for the general user; they have been provided mainly for those interested in developing new communication protocols, or for gaining access to some of the more cryptic facilities of an existing protocol.

Types of Servers

There are two types of servers you can have:

- **Iterative Server:** This is the simplest form of server where a server process serves one client and after completing the first request, it takes request from another client. Meanwhile, another client keeps waiting.
- **Concurrent Servers:** This type of server runs multiple concurrent processes to serve many requests at a time because one process may take longer and another client cannot wait for so long. The simplest way to write a concurrent server under Unix is to fork a child process to handle each client separately.

How to implement a client

The system calls for establishing a connection are somewhat different for the client and the server, but both involve the basic construct of a socket. Both the processes establish their own sockets. The steps involved in establishing a socket on the client side are as follows:

- Create a socket with the *socket()* system call.
- Connect the socket to the address of the server using the *connect()* system call.
- Send and receive data. There are a number of ways to do this, but the simplest way is to use the *read()* and *write()* system calls.

How to implement a Server Program in C using Linux APIs?

The steps involved in establishing a socket on the server side are as follows:

- Create a socket with the *socket()* system call.
- Bind the socket to an address using the *bind()* system call. For a server socket on the Internet, an address consists of a port number on the host machine.
- Listen for connections with the *listen()* system call.
- Accept a connection with the *accept()* system call. This call typically blocks the connection until a client connects with the server.
- Send and receive data using the *read()* and *write()* system calls.

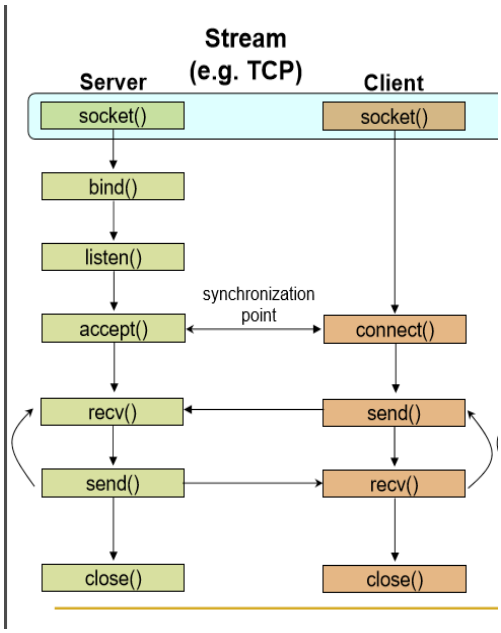


Figure 1.1 TCP client server interactions

Basic data structures used in Socket programming:

Various structures are used in Unix Socket Programming to hold information about the address and port, and other information. Most socket functions require a pointer to a socket address structure as an argument. Structures defined in this chapter are related to Internet Protocol Family.

- **Socket Descriptor**
  - A simple file descriptor in Unix. Data type is integer.
- **Socket Address**
  - This construct holds the information for socket address.

Table 1.1 System calls used in socket programming

Primitive	Meaning
Socket	Create a new communication endpoint
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive	Receive some data over the connection
Close	Release the connection

syntax

```
struct sockaddrs {
unsigned short sa_family; // address family, AF_xxx or //PF_xxx
char sa_data[14];              // 14 bytes of protocol address
 };
```

- AF stands for Address Family and PF stands for Protocol Family.

Table 2.2 Address Family

Name	**Purpose**
AF_UNIX, AF_LOCAL	Local communication
AF_INET	IPv4 Internet protocols
AF_INET6	IPv6 Internet protocols
AF_IPX	IPX - Novell protocols

- **struct sockaddr_in**
  - This construct holds the information about the address family, port number, Internet address,and the size of the struct sockaddr.
- struct sockaddr_in

```
{
    short int sin_family; // Address family unsigned short int sin_port;  // Port number
    struct in_addr sin_addr; // Internet address
 };
```

- The IP address structure, in_addr, is defined as follows

```
struct in_addr
{
  unsigned long int s_addr;
};
```

Some of the System Calls Used for Conversion

Some systems (like x8086) are Little Endian i.e., least significant byte is stored in the higher address, whereas in Big-Endian systems most significant byte is stored in the higher address. Consider a situation where a Little-Endian system wants to communicate with a Big Endian one, if there is no standard for data representation then the data sent by one machine is misinterpreted by the other. So the standard has been defined for the data representation in the network (called Network Byte Order) which is the Big Endian.

The system calls that help us to convert a short/long from Host Byte order to Network Byte Order and vice versa are:

- htons() -- "Host to Network Short"
- htonl() -- "Host to Network Long"
- ntohs() -- "Network to Host Short"
- ntohl() -- "Network to Host Long"

To ensure correct byte ordering of the 16-bit port number, your server and client need to apply these functions to the port address.

For example

```
server_address.sin_addr.s_addr= htonl(INADDR_ANY);
server_address.sin_port = htons(9734);
```

IP address is a 32bit integer-not convenient for humans. So, the address is written in dotted decimal representation.

- **inet_addr()** converts the Internet host address from the standard numbers-and-dots notation into binary data. It returns nonzero if the address is valid, zero if not.
- **inet_aton()** is also used for same purpose.

System calls used.

1. Socket creation in C using *socket()*

int sockid = socket(family, type, protocol);

- *sockid* is socket descriptor, an integer (like a file-handle)
- *family* is the communication domain, like PF_INET for IPv4 protocols and Internet addresses or PF_UNIX for Local communication and File addresses.
- *Type* defines communication type such as SOCK_STREAM or SOCK_DGRAM.
- *protocol* specifies protocol used. It take values like IPPROTO_TCP or IPPROTO_UDP but usually set to 0 (i.e., use default protocol).

If the return value *sockid* is negative values, it means there is problem in socket creation.
NOTE: socket call does not specify where data will be coming from, nor where it will be going to – it just creates the interface!

2. Assign address to socket using *bind()*

bind() associates and reserves a port for use by the socket.

Syntax:

int status = bind(sockid, &addrport, size);

- *Sockid* is a integer describing socket descriptor
- *addrport* is struct sockaddr which contains the (IP) address and port of the machine „ for TCP/IP server, internet address is usually set to INADDR_ANY, i.e., chooses any incoming interface
- *size* specifies the size (in bytes) of the addrport structure
- *Status* will be assigned -1 returns on failure.

3. Listening to connection requests using *listen()*

This system call instructs TCP protocol implementation to listen for connections

Syntax:

int status = listen(sockid, queueLimit);

- *Sockid* is socket descriptor which is created using *socket()*
- *Queuelemit* is an integer which specifies number of active participants that can "wait" for a connection
- *Status* will be assigned -1when returns on failure.

Note: The listening socket (sockid) is never used for sending and receiving. It is used by the server only as a way to get new sockets.

4. Establish Connection using *connect()*

The client establishes a connection with the server by calling *connect()*

Syntax:

int status = connect(sockid, &foreignAddr, addrlen);

- *sockid* is socket descriptor to be used in connection
- *foreignAddr is* struct sockaddr which contains address of the passive participant
- *addrlen is* sizeof(*foreignAddr*)

- *Status* will be assigned -1 when returns on failure

Note: *connect()* is blocking where as *listen()* is non blocking.

5. Accept incoming Connection using *accept()*

The server gets a socket for an incoming client connection by calling accept()

Syntax:

int newsockid = accept(sockid, &clientAddr, &addrLen);

- *newsockid* is an_integer, the new socket is created in server which is client specific and this new socket is used for data-transfer between server and client.
- *Sockid* is the socket created using socket system call, which is used only to listen to incoming requests from clients.
- *clientAdd*r is in the form of *struct sockaddr*, address of the active participant.
- *addrLen* is size of *clientAddr* parameter.

Note: *accept()* is blocking, it waits for connection before returning and dequeues the next connection on the queue for socket (sockid).

6. Exchanging data with stream socket

Application running in server and client(s) can transfer data using *send()* and *receive()* system call.

Syntax:

int count = send(sockid, msg, msgLen, flags);

- *Sockid* is the new socket descriptor created by accept in server side and socket in client side, depending on where it is used.
- *Msg* is an array holding message to be transmitted.
- *msgLen* holds length of message (in bytes) to transmit.
- flags are integer, special options, usually set 0
- Return value *count* has number of bytes transmitted and is set to -1 on error

Syntax:

int count = recv(sockid, recvBuf, bufLen, flags);

- *recvBuf* stores received message
- *bufLen* holds number if bytes

7. Closing the socket using *close()*

When finished using a socket, the socket should be closed.

Syntax:

int status= close(sockid);

- *sockid*: the file descriptor (socket being closed)
- status: 0 if successful, -1 if error

Closing a socket closes a connection (for stream socket) and frees up the port used by the socket.

II. SOLVED EXERCISE:

Write an iterative TCP client server program where client sends a message to server and server echoes back the message to client. Client should display the original message and echoed message.

Note: As socket is also a file descriptor, we can use read and write system calls to receive and send data.

Program:

Server code:

// Make the necessary includes and set up the variables:

```
#include<stdio.h>
#include<string.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#define PORTNO 10200
int main()
{
        int sockfd,newsockfd,portno,clilen,n=1;
        struct sockaddr_in seraddr,cliaddr;
        int i,value;
        // create an unnamed socket for the server
        sockfd = socket(AF_INET,SOCK_STREAM,0);
        //Name the socket
        seraddr.sin_family = AF_INET;
        seraddr.sin_addr.s_addr = inet_addr("172.16.59.10");// **
        seraddr.sin_port = htons(PORTNO);
        bind(sockfd,(struct sockaddr *)&seraddr,sizeof(seraddr));

        //Create a connection queue and wait for clients
        listen(sockfd,5);
```

```
        while (1) {
        char buf[256];
        printf("server waiting");
        //Accept a connection
        clilen = sizeof(clilen);
        newsockfd=accept(sockfd,(struct sockaddr *)&cliaddr,&clilen);
        //Read and write to client on client_sockfd (Logic for problem mentioned here)
        n = read(newsockfd,buf,sizeof(buf));
        printf(" \nMessage from Client %s \n",buf);
        n = write(newsockfd,buf,sizeof(buf));
          }
  }
```

- indicates replace this address with your systems IP address

Client Code:

```
//Make the necessary includes and set up the variables
#include<sys/types.h>
#include<sys/socket.h>
#include<stdio.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<stdlib.h>
#include<string.h>
int main()
{
      int len,result,sockfd,n=1;
      struct sockaddr_in address;
      char ch[256],buf[256];

      //Create a socket for the client
      sockfd = socket(AF_INET, SOCK_STREAM, 0);
```

```
//Name the socket as agreed with the server
address.sin_family=AF_INET;
address.sin_addr.s_addr=inet_addr("172.16.59.10");  **
address.sin_port=htons(10200);
len = sizeof(address);
//Connect your socket to the server's socket
result=connect(sockfd,(struct sockaddr *)&address,len);
if(result==-1)
{
        perror("\nCLIENT ERROR");
       exit(1);
}
//You can now read and write via sockfd (Logic for problem mentioned here)
printf("\nENTER STRING\t");
gets(ch);
ch[strlen(ch)]='\0';
write(sockfd,ch,strlen(ch));
printf("STRING SENT BACK FROM SERVER IS .....");
while(n){
n=read(sockfd,buf,sizeof(buf));
puts(buf);
}
}
```

- indicates replace this address with your systems IP address

Steps to execute the program.

1. Open two terminal windows and open a text file from each terminal with .c extension using command:

$gedit filename.c

2. Type the client and server program in separate text files and save it before exiting the text window.

3. First compile and run the server using commands mentioned below

a. $***gcc filename –o executablefileName //renaming the a.out file***

b. $./ executablefileName

4. Compile and run the client using the same instructions as listed in 3a & 3b.

Note: The ephemeral port number has to be changed every time the program is executed.

III. Lab Exercises:

Write **iterative** TCP client server 'C' programs to:

1. To illustrate encryption and decryption of messages using TCP. The client accepts messages to be encrypted through standard input device. The client will encrypt the string by adding 4(random value) to ASCII value of each alphabet. The encrypted message is sent to the server. The server then decrypts the message and displays both encrypted and decrypted forms of the string. Program terminates after one session.
2. Where the client accepts a sentence from the user and sends it to the server. The server will check for duplicate words in the string. Server will find number of occurrences of duplicate words present and remove the duplicate words by retaining single occurrence of the word and send the resultant sentence to the client. The client displays the received data on the client screen. The process repeats until the user enter the string "Stop". Then boththe processes terminate.

LAB NO: 2 **Date:**

Socket Programming in 'C' using TCP- Concurrent Client-Server Programs

Objectives:

- To implement concurrent servers to handle multiple requests by client at a time.

Prerequisites:

- Understanding of Multiprocessing/Multitasking Concepts of Operating Systems.
- Knowledge of the C programing language and Linux Networking APIs

I. Introduction to Socket Programming in 'C' using TCP/IP – Concurrent Servers

There might be a need to consider the case of multiple, simultaneous clients connecting to a server. The fact that the original socket is still available and that sockets behave as file descriptors gives you a method of serving multiple clients at the same time. If the server calls fork to create a second copy of itself, the open socket will be inherited by the new child process. It can then communicate with the connecting client while the main server continues to accept further client connections. This is, in fact, a fairly easy change to make to your server program, which is shown in the following.

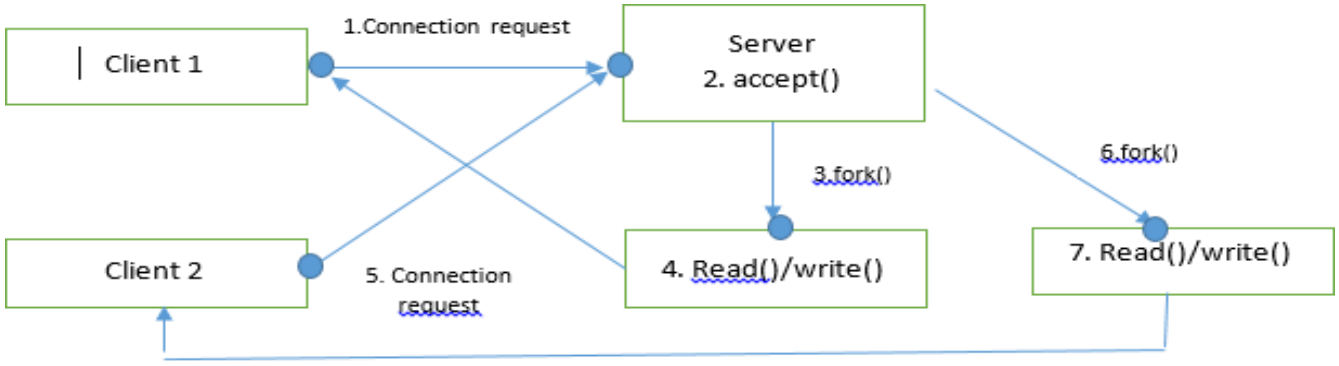


Figure 3.1 TCP concurrent server and clients interactions

Function Description: fork() The fork command creates a new separate process for each client. The fork() command splits the current process into two processes: a parent and a child. The new process (child process) is an almost exact copy of the process that calls it (the parent process). The fork() command returns 0 when called in the child process, returns the process ID of the newly created (child) process when called in the parent process, and –1 on error. Therefore, the return value of the function call to fork() tells the process whether it is the parent or the child. For a parent to keep track of its children, it should record the return values from call to fork(). (Note: If it is desired to get the process ID of the parent, the child can obtain it by calling getppid command.) From this point one can easily program the child process to serve the client's request while the parent can keep accepting other requests. However, when a child finishes and exits it needs to notify the parent that it is done

This is where the waitpid() command comes to screen.(Please do a man page on this function to learn more about it.)

We have seen how *fork()*can be used to handle multiple clients. But forking a new process is expensive, it duplicates the entire state (memory, stack, file/socket descriptors …).Threads decrease this cost by allowing multitasking within the same process. Both process and thread incurs overhead as they include creation, scheduling and context switching and also as their numbers increases, this overhead increases.

Example : fork()

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int main(void) {
  pid_t pid = fork();

  if(pid == 0) {
    printf("Child => PPID: %d PID: %d\n", getppid(), getpid());
    exit(EXIT_SUCCESS);
  }
  else if(pid > 0) {
    printf("Parent => PID: %d\n", getpid());
    printf("Waiting for child process to finish.\n");
    wait(NULL);
    printf("Child process finished.\n");
}
  else {
    printf("Unable to create child process.\n");
  }

  return EXIT_SUCCESS;
}
```

Output:

```
Parent => PID: 229
Waiting for the child process to finish.
Child => PPID: 229 PID: 230
Child process finished.
```

I. SOLVED EXERCISE ON CONCURRENT SERVERS

Write a TCP concurrent Echo server and simple client.

Server Program

```
#include<stdio.h>
#include<string.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#define PORTNO 10200
int main()
{
        int sockfd,newsockfd,portno,clilen,n=1;
        char buf[256];
        struct sockaddr_in seraddr,cliaddr;
        int i,value;
        sockfd = socket(AF_INET,SOCK_STREAM,0);
        seraddr.sin_family = AF_INET;
        seraddr.sin_addr.s_addr = inet_addr("172.16.59.10"); //**
        seraddr.sin_port = htons(PORTNO);
        bind(sockfd,(struct sockaddr *)&seraddr,sizeof(seraddr));
        // Create a connection queue, ignore child exit details, and wait for clients
        listen(sockfd,5);
        while(1){
                //Accept the connection
                clilen = sizeof(clilen);
                newsockfd=accept(sockfd,(struct sockaddr *)&cliaddr,&clilen);
               //Fork to create a process for this client and perform a test to see whether
               //you're the parent or the child:
                if(fork()==0){
                    // If you're the child, you can now read/write to the client on newsockfd.
                     n = read(newsockfd,buf,sizeof(buf));
```

```
                printf(" \nMessage from Client %s \n",buf);
                n = write(newsockfd,buf,sizeof(buf));
                close(newsockfd);
                exit(0);
              }
          //Otherwise, you must be the parent and your work for this client is finished
        else
            close(newsockfd);
      }
}
```

- indicates replace this address with your systems IP address

Client Program remains same.

Steps for execution

1. Open three terminals
2. Terminal 1: $gcc server.c –o server
   ./server
3. Terminal 2 : $gcc client.c –o client1
   ./client1
4. Terminal 3: $gcc client.c –o client2
   ./client2

2. LAB EXERCISES

1. Write a TCP concurrent client server program where server accepts integer array from client and sorts it and returns it to the client along with process id.
2. Implement concurrent Remote Math Server To perform arithmetic operations in the server and display the result to the client. The client accepts two integers and an operator from the user and sends it to the server. The server then receives integers and operator. The server will perform the operation on integers and sends the result back to the client which is displayed on the client screen. Then both the processes terminate.

3. Implement simple TCP daytime server using select().

3. Lab Exercises:

1. Write a concurrent TCP daytime server 'C' program. Along with the result, server should also send the process id to the client.
2. Write a concurrent TCP client-server 'C' program where the client accepts a sentence from the user and sends it to the server. The server will check for duplicate words in the string. Server will find the number of occurrences of duplicate words present and remove the duplicate words by retaining single occurrence of the word and send the resultant sentence to the client. The client displays the received data on the client screen. The process repeats until the user enters the string "Stop". Then both processes terminate.

Additional Exercises:

Q2.3) Write a concurrent TCP daytime server 'C' program. Along with the result, server should also send the process id to client.

Q2.4) Write a concurrent TCP client server 'C' program where the client accepts a sentence from the user and sends it to the server. The server will check for duplicate words in the string. Server will find the number of occurrences of duplicate words present and remove the duplicate words by retaining single occurrence of the word and send the resultant sentence to the client.

LAB NO: 3 **Date:**

Socket Programming in 'C' using UDP and Network Monitoring and Analysis with Wireshark

Objectives:

- Understand UDP Client-Server Socket-Programming
- To monitor network data using Wireshark tool.

Perquisites:

- Understanding of connectionless service.

I. Introduction to User Datagram Protocol

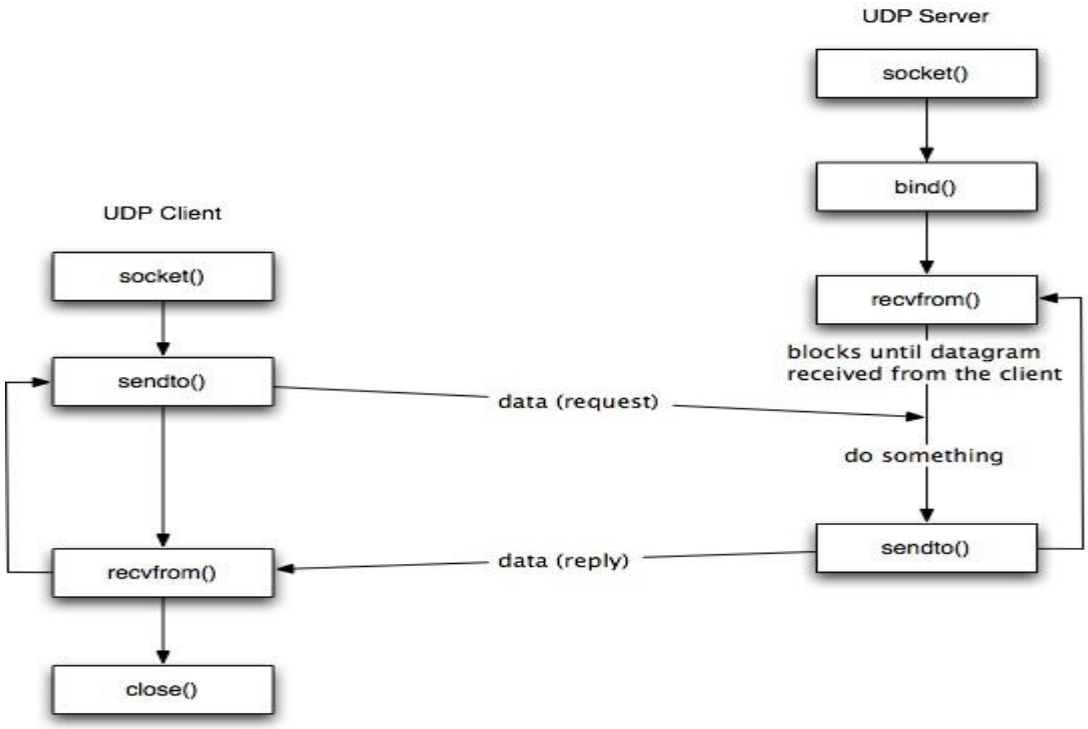


Figure 3.1 Interaction between UDP client and Server.

System calls used in UDP:

1. Socket creation using socket() : Only difference in this call is its second parameter. For TCP, we use stream sockets as data is transferred in the form of the stream whereas in UDP, data is transmitted in the form of datagrams. So, the type of socket used for UDP transmission is DATAGRAM socket. The system call should be used as follows.

   sockfd = socket(AF_INET, SOCK_DGRAM, 0);

2. Exchange of data using sendto() and recvfrom(): As UDP does not establish the connection, while sending datagram, it should specify the address of the receiver.
   Syntax:

   int sendto(int sockfd, const void *msg, int len, unsigned int flags,const struct sockaddr *to, int tolen);

- *sockfd*: It is a socket descriptor returned by the socket function.
- *msg:* It is a pointer to the data you want to send.
- *len*: It is the length of the data you want to send (in bytes).
- *flags*: It is set to 0.
- *to*: It is a pointer to struct sockaddr for the host where data has to be sent.
- *tolen*: It is set it to sizeof(struct sockaddr).
- Return value will be number of bytes sent or -1 for error.
  Similarly for receiving data, recvfrom is used and syntax is as follows
  Syntax:
  int recvfrom(int sockfd, void *buf, int len, unsigned int flags,struct sockaddr *from, int *fromlen);
- *sockfd*: It is a socket descriptor returned by the socket function.
- *buf*: It is the buffer to read the information into.
- *len*: It is the maximum length of the buffer.
- *flags*: It is set to 0.
- *from*: It is a pointer to struct sockaddr for the host where data has to be
- read.
- *fromlen*: It is set it to sizeof(struct sockaddr).
- Return value will be number of bytes recieved or -1 for error.

INTRODUCTION TO WIRESHARK

Wireshark is a network packet analyzer. A network packet analyzer will try to capture network packets and tries to display that packet data as detailed as possible.
Purposes:

- Network administrators use it to troubleshoot network problems.
- Network security engineers use it to examine security problems.
- Developers use it to debug protocol implementations.
- People use it to learn network protocol internals.

Getting started with Wireshark:

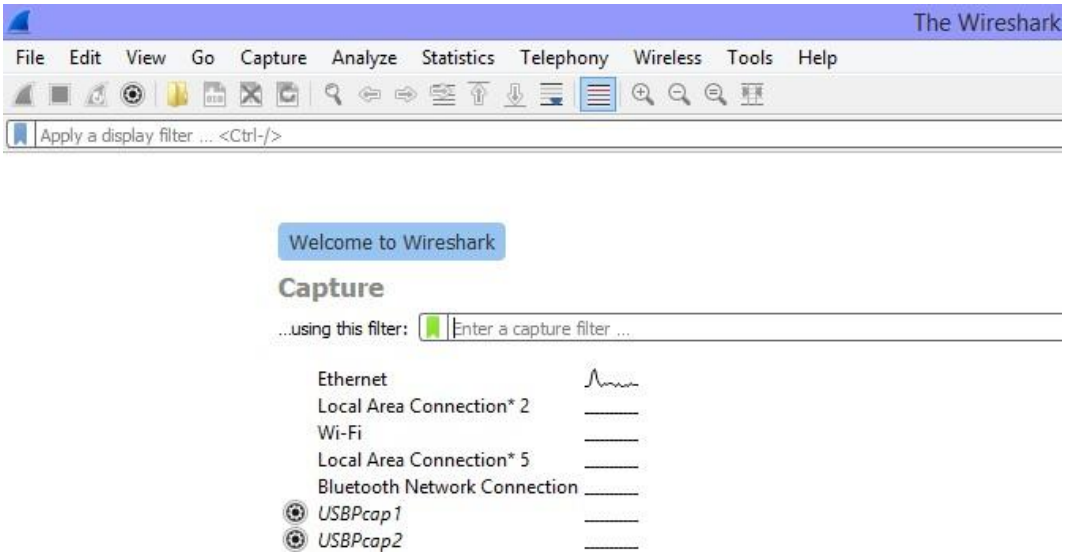


Figure 4.2: Welcome screen of Wireshark

- Choose the interface as "Ethernet". Once an interface is selected, if the computer is connected to the network and if there is some network traffic, the main window will be displayed. Click on the "start capturing packets" button on the main toolbar.

The capture is split into 3 parts:
1. Packet List Panel – this is a list of packets in the current capture. It colors the packets based on the protocol type. When a packet is selected, the details are shown in the two panels below.
2. Packet Details Panel – this shows the details of the selected packet. It shows the different protocols making up the layers of data for this packet. Layers include Frame, Ethernet, IP, TCP/UDP/ICMP, and application protocols such as HTTP.
3. Packet Bytes Panel – shows the packet bytes in Hex and ASCII encodings.

Wireshark uses two types of filters, capture filters and display filters. Capture filters are used to decide which packets should be kept. Only packets that meet filter criteria will be kept.

Display filters work after the capture is completed. They restrict which packets are shown, but they don't discard any information. Capture filters would be more useful on very busy networks when you need to limit the amount of data your machine needs to process. On the other hand, display filters don't save any memory; display filters let you temporarily focus on an analysis without losing any underlying information.

Capture filters can be set in two different places. Go to the Capture menu and select "Options" and you will find a selection for capture filters. Alternatively, go to the Capture menu and select "Capture Filters". From the "Capture Filters" dialog box you will see a help menu that will explain how the function works. Display filters can be entered at the top of the display screen.

Display filters can be set as: Right click on the Source IP address field in the Packet Details Panel. Select Prepare a Filter->Selected. Wireshark automatically generates a Display Filter and applies it to the capture. The filter is shown in the Filter Bar, below the button toolbar. Only packets captured with a Source IP address of the value selected should be displayed. This same process can be performed on most fields within Wireshark and can be used to include or exclude traffic.

SOLVED EXERCISE

Write a UDP Echo client server program.

Server Program:

```
#include<stdio.h>
#include<fcntl.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<unistd.h>
int main()
{
        int sd;
        char buf[25];
        struct sockaddr_in sadd,cadd;
```

```
        //Create a UDP socket
          sd=socket(AF_INET,SOCK_DGRAM,0);

        //Construct the address for use with sendto/recvfrom...  */
          sadd.sin_family=AF_INET;
          sadd.sin_addr.s_addr=inet_addr("172.16.56.10");//**
          sadd.sin_port=htons(9704);
          int result=bind(sd,(struct sockaddr *)&sadd,sizeof(sadd));
          int len=sizeof(cadd);
          int m=recvfrom(sd,buf,sizeof(buf),0,(struct sockaddr *)&cadd,&len);
          printf("the server send is\n");
          puts(buf);
          int n=sendto(sd,buf,sizeof(buf),0,(struct sockaddr *)&cadd,len);
          return 0;
}
```

- indicates replace this address with your systems IP address

Client program

```
#include<stdio.h>
#include<stdlib.h>
#include<fcntl.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<unistd.h>
int main()
{
        int sd;
        struct sockaddr_in address;
        sd=socket(AF_INET,SOCK_DGRAM,0);
        address.sin_family=AF_INET;
        address.sin_addr.s_addr=inet_addr("172.16.56.10");//**
        address.sin_port=htons(9704);
```

```
    char buf[25],buf1[25];

    printf("enter buf\n");
    gets(buf);
    int len=sizeof(address);
    int m=sendto(sd,buf,sizeof(buf),0,(struct sockaddr *)&address, len);
    int n=recvfrom(sd,buf,sizeof(buf),0,(struct sockaddr *)&address,&len);
    printf("the server echo is\n");
    puts(buf);
    return 0;
}
```

****- indicates replace this address with your systems IP address***

LAB EXERCISES:

1. Write a UDP client-server program where the client sends rows of a matrix, and the servercombines them together as a matrix.
2. Write a client program to send a manually crafted HTTP request packet to a Web Server and display all fields received in HTTP Response at client Side.

2. Analyzing UDP datagrams using Wireshark:

- Start your web browser and clear the browser's cache memory, but do not access any website yet.
- Open Wireshark and start capturing.
- Go back to your web browser and retrieve any file from a website. Wireshark starts capturing packets.
- After enough packets have been captured, stop Wireshark, and save the captured file.
- Using the captured file, analyze TCP & UDP packets captured
  Note: DNS uses UDP for name resolution & HTTP uses TCP

Using the captured information, answer the following questions in your lab report.

A. In the packet list pane, select the first DNS packet. In the packet detail pane, select the **User Datagram Protocol**. The UDP hexdump will be highlighted in the packet byte lane. Using the hexdump, Answer the following:

a. the source port number.

b. the destination port number.

c. the total length of the user datagram.

d. the length of the data.

e. whether the packet is directed from a client to a server or vice versa.

f. the application-layer protocol.

g. whether a checksum is calculated for this packet or not.

B. What are the source and destination IP addresses in the DNS query message? What are those addresses in the response message? What is the relationship between the two?

C. What are the source and destination port numbers in the query message? What are those addresses in the response message? What is the relationship between the two? Which port number is a well-known port number?

D. What is the length of the first packet? How many bytes of payload are carried by the first packet?

2b. Analyzing TCP packets using Wireshark:

Start your web browser and clear the browser's cache memory, but do not access any website yet.

■ Open Wireshark and start capturing.

■ Go back to your web browser and retrieve any file from a website. Wireshark starts capturing packets.

■ After enough packets have been captured, stop Wireshark and save the captured file.

■ Using the captured file, select only those packets that use the service of TCP. For this purpose, type **tcp** (lowercase) in the *filter field* and press **Apply**. The packet list pane of the Wireshark window should now display a bunch of packets.

Part I: Connection-Establishment Phase

Identify the TCP packets used for connection establishment. Note that the last packet used for connection establish may have the application-layer as the source protocol.

Questions

Using the captured information, answer the following question in your lab report about packets used for connection establishment.

1. What are the socket addresses for each packet?

2. What flags are set in each packet?

3. What are the sequence number and acknowledgment number of each packet?

4. What are the window size of each packet?

Part II: Data-Transfer Phase

The data-transfer phase starts with an HTTP GET request message and ends with an HTTP OK message.

Questions

Using the captured information, answer the following question in your lab report about packets used for data transfer.

1. What TCP flags are set in the first data-transfer packet (HTTP GET message)?

2. How many bytes are transmitted in this packet?

3. How often does the receiver generate an acknowledgment? To which acknowledgment rule (defined in Page 200 in the textbook) does your answer correspond to?

4. How many bytes are transmitted in each packet? How are the sequence and acknowledgment numbers related to number of bytes transmitted?

5. What are the original window sizes that are set by the client and the server? Are these numbers expected? How do they change as more segments are received by the client?

6. Explain how the window size is used in flow control?

7. What is the purpose of the HTTP OK message in the data transfer phase?

Part III: Connection Termination Phase

The data-transfer phase is followed by the connection termination phase. Note that some packets used in the connection-termination phase may have the source or sink protocol at the application layer. Find the packets used for connection termination.

Questions

Using the captured information, answer the following question in your lab report about packets used for connection termination.

1. How many TCP segments are exchanged for this phase?

2. Which end point started the connection termination phase?

3. What flags are set in each of segments used for connection termination?

II. LAB EXERCISES

1. From the captured information, answer the following question in your lab report.

   i. Using the hexdump, determine the following for any TCP packet:
   The source port number, the destination port number, the sequence number, the acknowledgment number, the header length, the set flags, the window size and the urgent pointer value.

   ii. Using the information in the detail pane lane, verify your answers is question 1.

   iii. Does any of the TCP packet header carry options? Explain your answer.

   iv. What is the size of a TCP packet with no options. What is the size of a TCP packet with options?

   v. Is window size in any of the TCP packet zero? Explain your answer.

2. Analyze Interaction between your TCP Client-Server Programs using wireshark
3. Analyze Interaction between your UDP Client-Server Programs using wireshark

LAB NO: 4 **Date:**

Network Data Analysis using tcpdump

Objectives

- Understanding the network analysis tool - tcpdump

Network Analysis tool

Network Analysis tools are used to identify problems in the network, as well as to help understand the behavior of network protocols. We will use these tools extensively in the experiments.

Tcpdump:

Tcpdump is a network traffic sniffer built on the packet capture library libpcap. While started, it captures and displays packets on the LAN segment. By analyzing the traffic flows and the packet header fields, a great deal of information can be gained about the behavior of the protocols and their operation within the network. Problems in the network can also be identified. A packet filter can be defined in the command line with different options to obtain a desired output.

Basics

Below are a few options you can use when configuring tcpdump.

Options

- **-i any** : Listen on all interfaces just to see if you're seeing any traffic.
- **-I eth0** : Listen on the eth0 interface.
- **-D** : Show the list of available interfaces
- **-n** : Don't resolve hostnames.
- **-nn** : Don't resolve hostnames *or* port names.
- **-q** : Be less verbose (more quiet) with your output.
- **-t** : Give human-readable timestamp output.
- **-tttt** : Give maximally human-readable timestamp output.
- **-X** : Show the packet's *contents* in both hex and ascii.
- **-XX** : Same as **-X**, but also shows the ethernet header.

- **-v, -vv, -vvv** : Increase the amount of packet information you get back.
- **-c** : Only get *x* number of packets and then stop.
- **-s** : Define the *snaplength* (size) of the capture in bytes. Use -s0 to get everything, unless you are intentionally capturing less.
- **-S** : Print absolute sequence numbers.
- **-e** : Get the ethernet header as well.
- **-q** : Show less protocol information.
- **-E** : Decrypt IPSEC traffic by providing an encryption key.

Expressions

In tcpdump, *Expressions* allow you to trim out various types of traffic and find exactly what you're looking for. Mastering the expressions and learning to combine them creatively is what makes one truly powerful with tcpdump.

There are three main types of expression: type, dir, and proto.

- Type options are: host, net, and port.
- Direction lets you do src, dst, and combinations thereof.
- Proto(col) lets you designate: tcp, udp, icmp, ah, and many more.

Examples

So, now that we've seen what our options are, let's look at some real-world examples that we're likely to see in our everyday work.

BASIC COMMUNICATION

By looking at all interfaces.

```
# tcpdump -i any
```

SPECIFIC INTERFACE

```
# tcpdump -i eth0

# tcpdump -ttttnnvvS
```

FIND TRAFFIC BY IP

One of the most common queries, this will show you traffic from 1.2.3.4, whether it's the source or the destination.

```
# tcpdump host 1.2.3.4
```

FILTERING BY SOURCE AND DESTINATION

```
# tcpdump src 2.3.4.5
# tcpdump dst 3.4.5.6
```

FINDING PACKETS BY NETWORK

To find packets going to or from a particular network, use the `net`option. You can combine this with the `src` or `dst` options as well.

```
# tcpdump net 1.2.3.0/24
```

SHOW TRAFFIC RELATED TO A SPECIFIC PORT

You can find specific port traffic by using the `port` option followed by the port number.

```
# tcpdump port 3389

# tcpdump src port 1025
```

SHOW TRAFFIC OF ONE PROTOCOL

```
# tcpdump icmp
```

SHOW ONLY IP6 TRAFFIC

```
# tcpdump ip6
```

FIND TRAFFIC USING PORT RANGES

You can also use a range of ports to fin traffic.

```
# tcpdump portrange 21-23
```

FIND TRAFFIC BASED ON PACKET SIZE

```
# tcpdump less 32

# tcpdump greater 64

# tcpdump <= 128
```

WRITING CAPTURES TO A FILE

It's often useful to save packet captures into a file for analysis in the future. These files are known as PCAP (PEE-cap) files, and they can be processed by hundreds of different applications, including network analyzers, intrusion detection systems, and of course by itself. Here we're writing to a file called *capture_file* using the switch.

```
# tcpdump port 80 -w capture_file
```

READING PCAP FILES

You can read PCAP files by using the `-r` switch. Note that you can use all the regular commands within tcpdump while reading in a file; you're only limited by the fact that you can't capture and process what doesn't exist in the file already.

```
# tcpdump -r capture_file
```

Advanced

FROM SPECIFIC IP AND DESTINED FOR A SPECIFIC PORT

Let's find all traffic from 10.5.2.3 going to any host on port 3389.

```
tcpdump -nnvvS src 10.5.2.3 and dst port 3389
```

I. LAB EXERCISES:

1. While **tcpdump host** *your_host* is running in one command window, run ping 127.0.0.1 from another command window. From the ping output, is the 127.0.0.1 interface on? Can you see any ICMP message sent from your host in the tcpdump output? Why?

2. While **tcpdump host** *your_host* is running to capture traffic from your machine, execute telnet 128.238.66.200. Note there is no host with this IP address in the current configuration of the lab network. Save the tcpdump output of the first few packets for the lab report. After getting the necessary output, terminate the telnet session. From the saved tcpdump output, describe how the ARP timeout and retransmission were performed. How many attempts were made to resolve a non-existing IP address?

3. Briefly explain the purposes of the following tcpdump expressions.

a. tcpdump udp port 520
b. tcpdump -x -s 120 ip proto 89
c. tcpdump -x -s 70 host ip addr1 and (ip addr2 or ip addr3)
d. tcpdump -x -s 70 host ip addr1 and not ip addr2

4. Basic packet decoding

1) Write a tcpdump command to dump network traffic from an Ethernet connection to the screen in human readable output format. Perform the following operation and write down the observations.

   a) Capture all the traffic of maximum snap length of 65,535 bytes and provide the hexadecimal and ASCII decodes of all the traffic in each packet.

   b) Find the IP addresses, IP packet length, TCP port numbers, TCP flags, etc. by using the reference chart to locate those fields on the hexadecimal dump.

LAB No 5 **Date:**

Computer Network Design using HUB in GNS3

Objectives:

- Study of network simulator GNS3.
- To Learn Static IP address Assignment.
- To study the characteristics of HUB devices.

Introduction to GNS3

GNS3 is a free graphical network simulator capable of emulating a number of network devices. Supported devices include Cisco routers and firewalls, Juniper routers, and frame-relay switches. With this software, users get an easy-to-use interface that allows them to build complexlabs consisting of a variety of supported Cisco routers. GNS3 works by using real Cisco IOS images which are emulated using a program called Dynamips.

Some Supported GNS3 Features

- Design of high quality and complex network topologies
- Emulation of many Cisco router platforms and PIX firewalls
- Simulation of simple Ethernet, ATM and Frame Relay switches
- Connection of the simulated network to the real world
- Packet capture using Wireshark.

II. Screen Layout

The following figure shows a screenshot of the GNS graphical user interface:

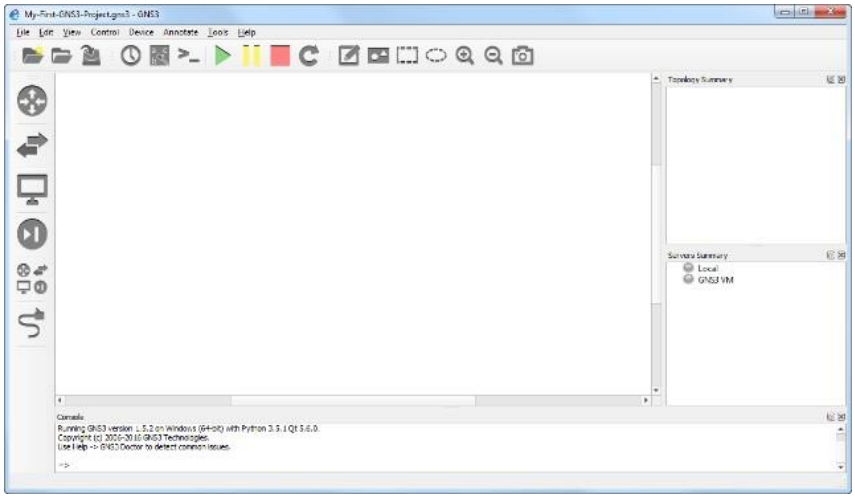


Figure 5.1 GNS graphical user interface

GNS3 Workspace: The GNS3 workspace is the area of GNS3 where you create topologies by adding devices and links.

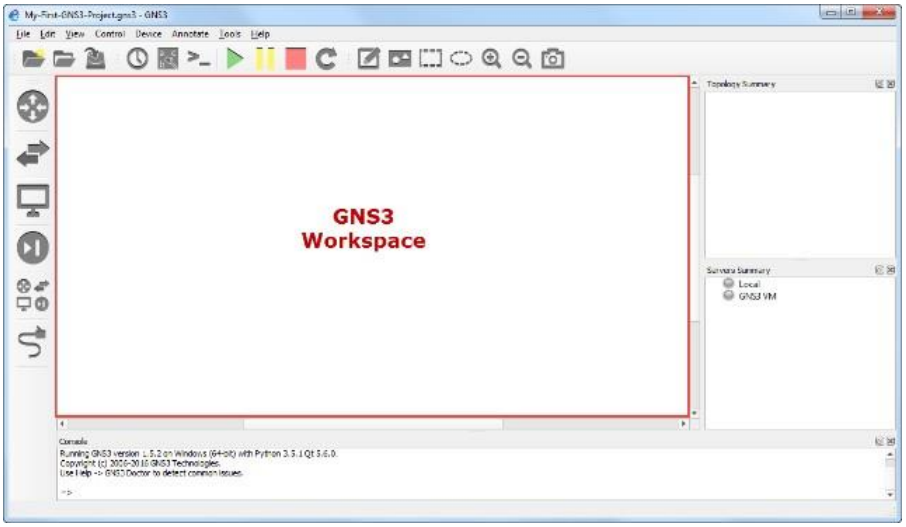


Figure 5.2 GNS workspace

Devices Toolbar: The devices toolbar allows you to add devices to your network topology. You do this by dragging devices from the Toolbar to the GNS3 workspace.

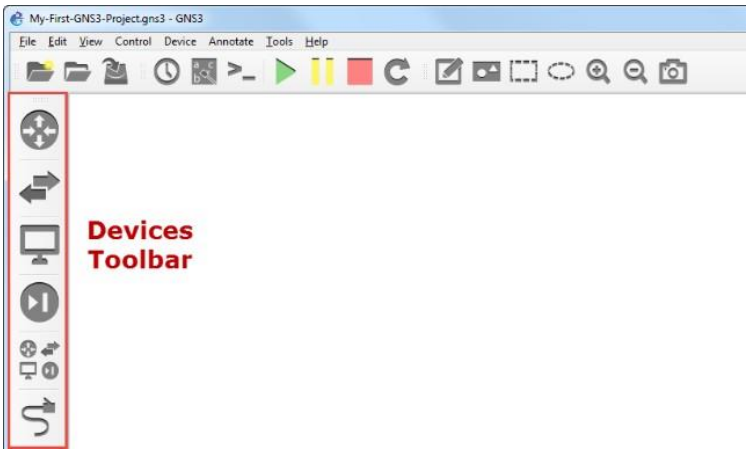


Figure 5.3 GNS Devices Toolbar

The devices toolbar is grouped into different types by default:

Routers:

GNS3 requires one or more Cisco IOS images to run on your virtual Dynamips routers, and GNS3 does not provide them. Images can be copied from a router you own or through a Cisco

connection online (CCO) account if you have a contract with Cisco. For the lab purpose, we have downloaded Cisco 3600 IOS image and Cisco 3700 IOS image from the net.

Figure 5.4 GNS Router

Switches:

Switching is done with only “Ethernet Switch” unless specified otherwise in the lab. When connecting to a switch select unused ports for new connections.

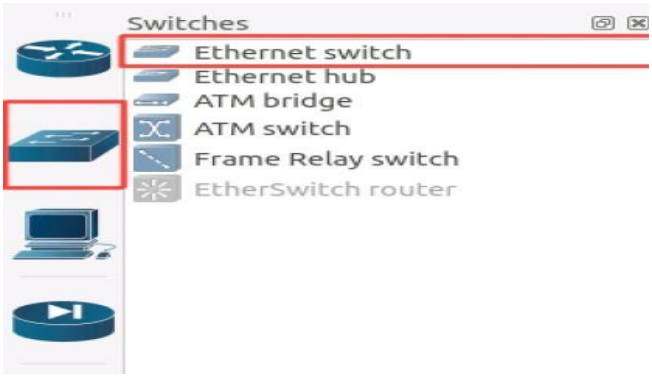


Figure 5.5 GNS Switch

Virtual PCs:

VPCS are lightweight Linux machine emulators. Each one runs within a single process, it has many limitations such as lacking a full Linux command list, as well as only having a single interface.

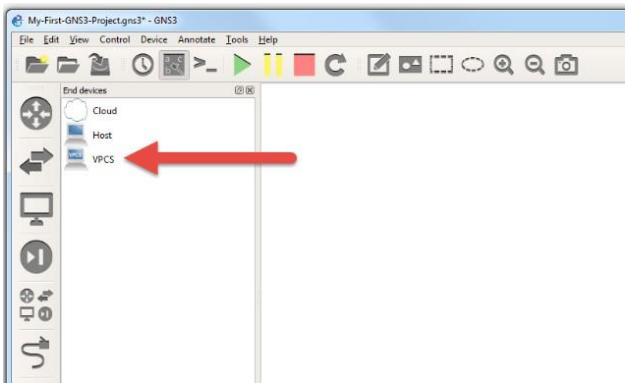


Figure 5.6 GNS VPCS

Add Links:

Adding links is simple.

1. Click the icon shown to the right.

2. Click on the target and select which port to attach a virtual connection to, do the same with the next target.

Figure 5.7 GNS Links

III. Using the GNS3 Toolbar

The GNS3 toolbar contains several groups of icons that are roughly organized by function and offer a simple way to get things done. The first group deals with projects, the second with links, the third with devices and snapshots, and the fourth with additional ways to visually organize your projects.

First Toolbar Group

The first group of toolbar icons, shown in Figure 5.8, deals with actions that affect entire projects.

From left to right, these icons are as follows:

New blank project Creates a new project folder and allows you to choose what to name your project.

Open project Opens a previously saved project. To open a project, choose the project folder name and select the file named <*project_name*>.*gns3*.

Save project Saves a complete project to the GNS3 *projects* folder. By default, a PNG image file of your workspace is saved with your project.

Figure 5.8: First toolbar group

Second Toolbar Group

The buttons in the second group of toolbar icons, shown in Figure 5.9, allow you to create project snapshots, show or hide interface labels, and connect to your devices using the virtual console port on your devices.

From left to right, these icons are as follows:

Snapshot Creates a snapshot of your devices, links, and IOS configurations to record the state of your workspace at that time. You can save more than one snapshot and revert to a saved snapshot at any time. Options are Create, Delete, Restore, and Close.

Show interface labels Shows or hides interface names used by a link. These labels are abbreviated and displayed with devices in your workspace (for example, f0/0 is displayed for FastEthernet0/0).

Console connect to all devices Opens a console connection to all running routers in your workspace.

Figure 5.9: Second toolbar group

Third Toolbar Group

The third group of toolbar icons, shown in Figure 5.10, primarily deals with controlling devices.

Figure 5.10: Third toolbar group

From left to right, these four icons are as follows:

Start/Resume all devices Starts all stopped devices or resumes all suspended devices in your workspace.

Suspend all devices Places all suspend-capable devices in a suspended state.

Stop all devices Stops all devices.

Reload all devices Reloads all devices. Be sure to save your router configurations and project before reloading or else you might lose your configurations.

Fourth Toolbar Group

The final group of toolbar icons, shown in Figure 5.11, provides tools to present your network layouts more clearly. You can add objects such as rectangles and ellipses to your project, and even generate a screenshot of your workspace.

Figure 5.11: Fourth toolbar group

From left to right, the icons in the last toolbar group are as follows:

Add a note Creates text annotations in your workspace. Double-click text to modify it, and right-click the text object to change the Style attributes (such as font size and color). You can also rotate text objects from 0 to 360 degrees.

Insert a picture Adds images and logos to your projects. GNS3 supports PNG, JPG, BMP, XPM, PPM, and TIFF file formats.

Draw a rectangle draws dynamically sizable rectangles. You can right click a rectangle object to change the Style attributes for border and border color. Rectangle objects can be rotated from 0 to 360 degrees.

Draw an ellipse draws dynamically sizable ellipses. You can right-click an ellipse object to change the border style and color.

Zoom in Zooms in your workspace to see details.

Zoom out Zooms out of your workspace for a bigger bird's-eye view.

Screenshot Generates a screenshot of your workspace. The image can be saved as a PNG, JPG, BMP, XPM, PPM, or TIFF file and by default is saved in your *GNS3/projects* folder.

Objects (notes, pictures, and shapes) that you add to your workspace can be grouped into layers. To raise or lower an object, right-click the object and select **Raise one layer** or **Lower one layer**. This feature allows you to manipulate objects in a layer without affecting other layers. You can display layer positions for your objects by choosing **View->Show Layers** from the menu, which is useful during advanced layer manipulation. By adding shapes and colors with this toolbar, you

can divide network components into logical groups. With text, you can add notes and reminders about how your project is configured.

IV. Address Resolution Protocol (ARP) and Reverse Address Resolution Protocol (RARP)

The ARP and RARP protocols perform the translation between IP addresses and MAC layer addresses. We will discuss ARP for broadcast LANs, particularly Ethernet LANs

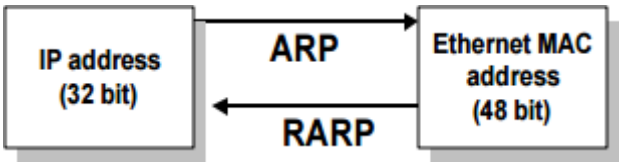


Figure 5.12: Information Exchange

- **ARP Packet**

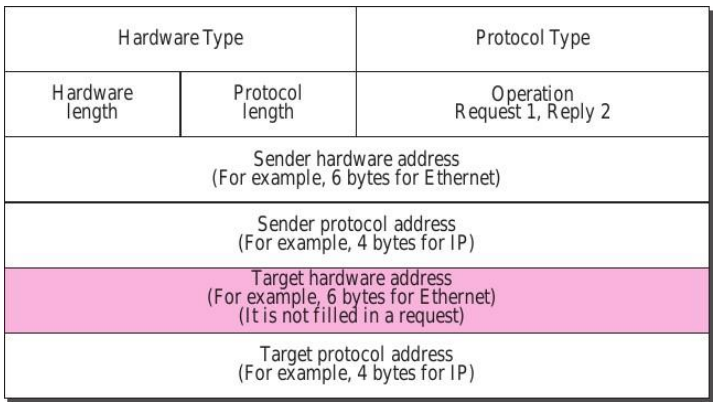


Fig. 5.13 ARP Packet

- **Encapsulation of ARP Packet**

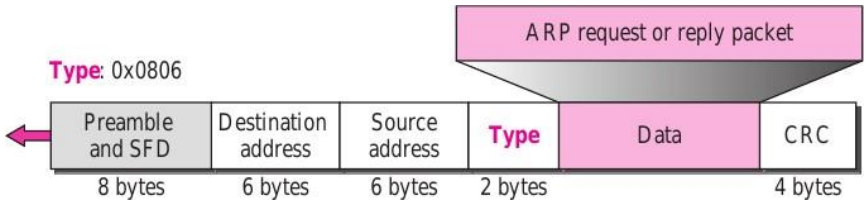


Fig. 5.14 ARP Packet

Illustration of Address Translation with ARP

ARP Request: Argon broadcasts an ARP request to all stations on the network: "What is the hardware address of Router137?"

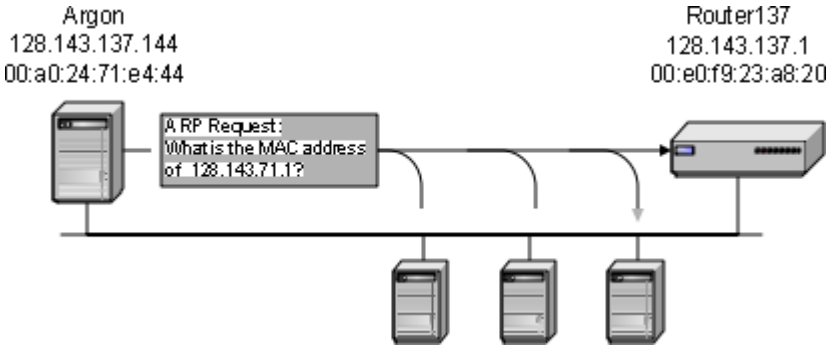


Figure 5.15:ARP Request

ARP Reply: Router 137 responds with an ARP Reply which contains the hardware address

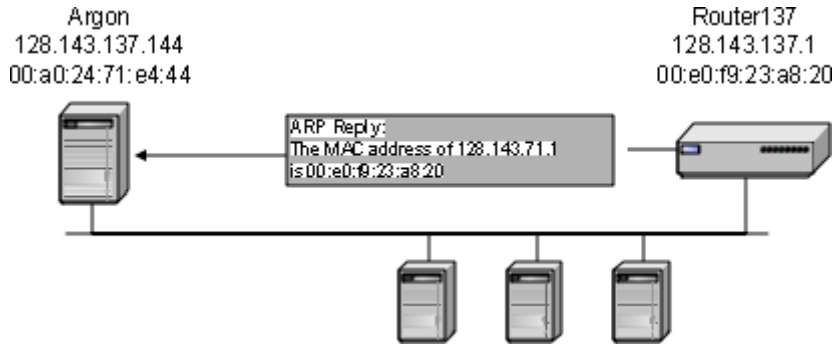


Figure 5.16: ARP Reply

Address Translation done for the above illustration:

ARP Request from Argon:

Source hardware address: 00:a0:24:71:e4:44

Source protocol address: 128.143.137.144

Target hardware address: 00:00:00:00:00:00

Target protocol address: 128.143.137.1

ARP Reply from Router137:

Source hardware address: 00:e0:f9:23:a8:20

Source protocol address: 128.143.137.1

Target hardware address: 00:a0:24:71:e4:44

Target protocol address: 128.143.137.144

V. SOLVED EXERCISE:

When you first start GNS3, you will be prompted to create a project

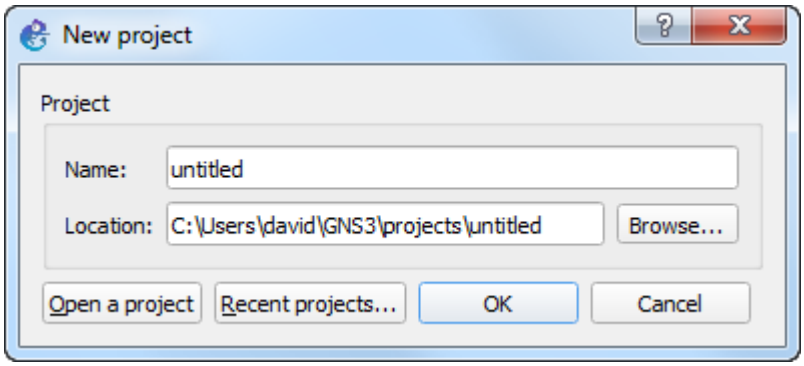


Figure 5.17:New Project Window

Name the project as desired and then click **OK**.

Working with VPCS

1. Drag and drop the **VPCS** node (device) to the GNS3 **Workspace**. An instance of the node becomes available in the **Workspace**. In this example a new VPCS with the name PC1 is now available:

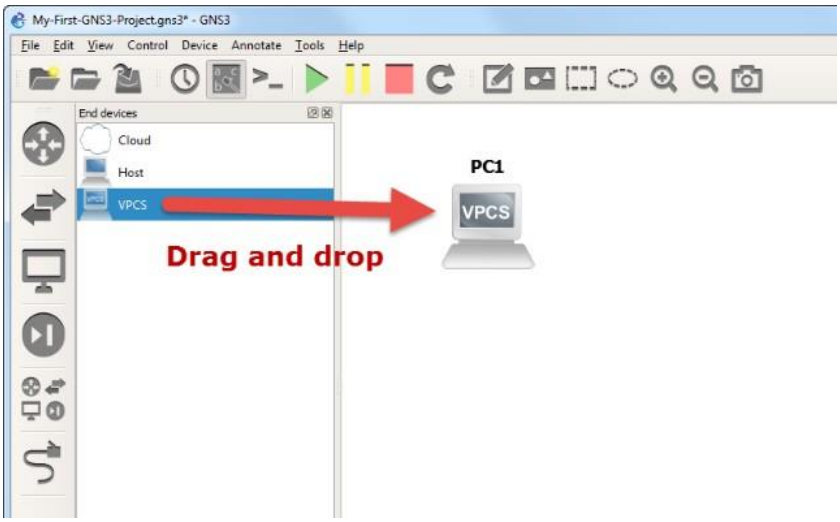


Figure 5.18: Dragging of First VPCS

2. Drag and drop the **VPCS** node again into the GNS3 **Workspace**. In this example, another VPCS was added to the GNS3 workspace (**PC2**):

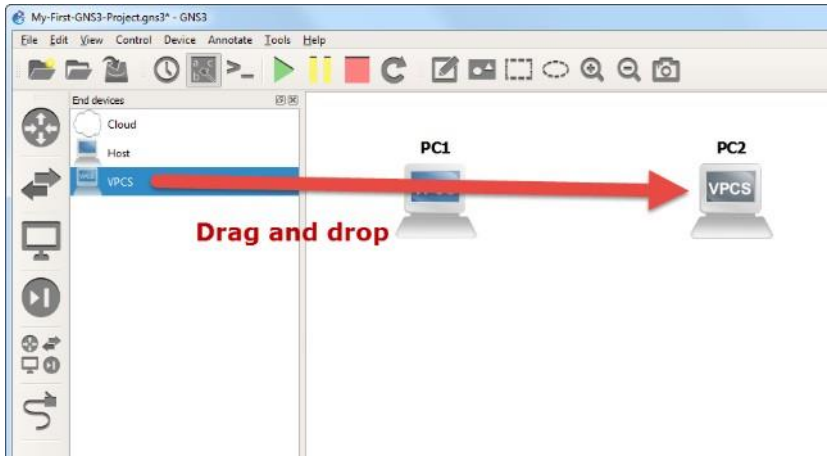


Figure 5.19:Dragging of Second VPCS

3. Click the **Add a Link** button to start adding links to your topology. The mouse cursor will change to indicate that links can be added:

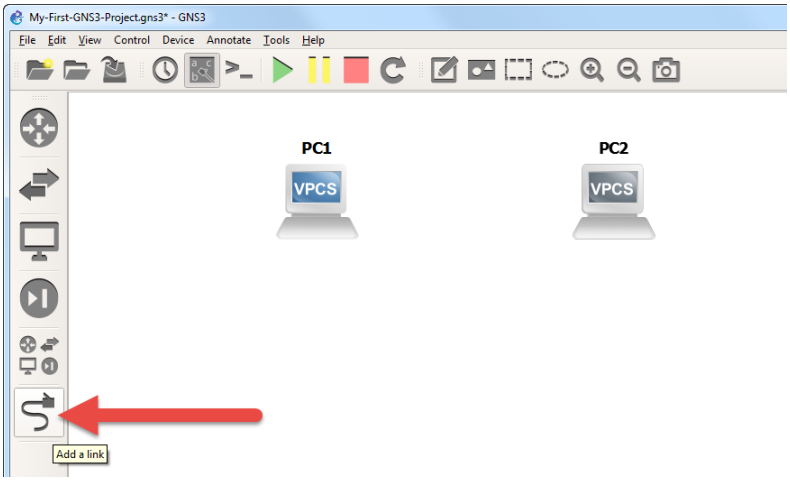


Figure 5.20: **Add a Link** button

4. Click on **PC1** in your topology to display available interfaces. In this example **Ethernet0** is available (this is device dependent):

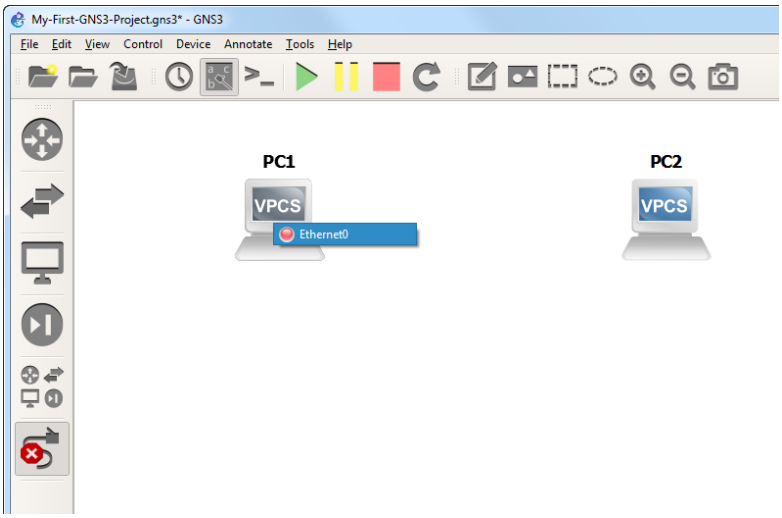
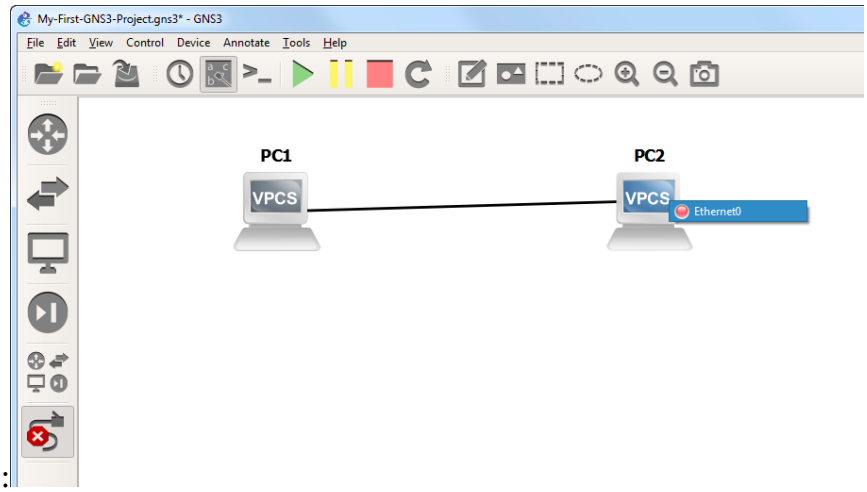


Figure 5.21: **Ethernet0** of PC0 is displayed

5. Click **Ethernet0** on **PC1** and select **PC2**



:

Figure 5.22: **Ethernet0** of PC1 is displayed

6. Select **Ethernet0** on **R2** to complete the connection:

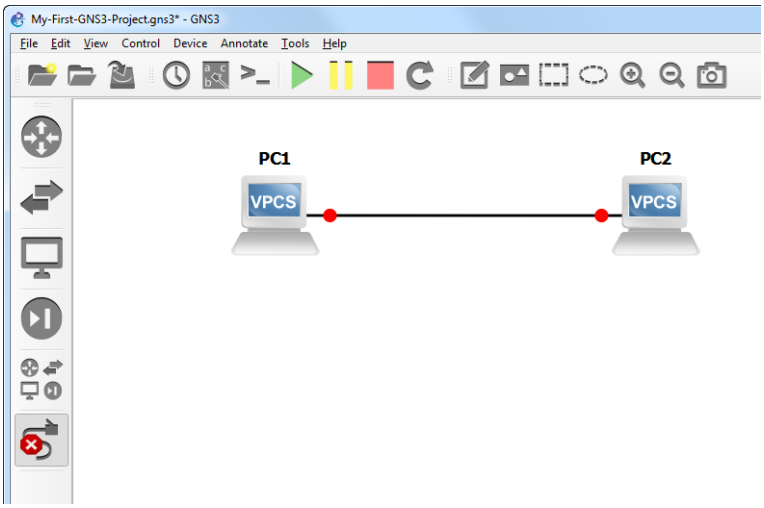


Figure 5.23: **Connection Complete**

7. Click the **Show/Hide interface labels** button on the **GNS3 Toolbar** to display interface labels in your topology

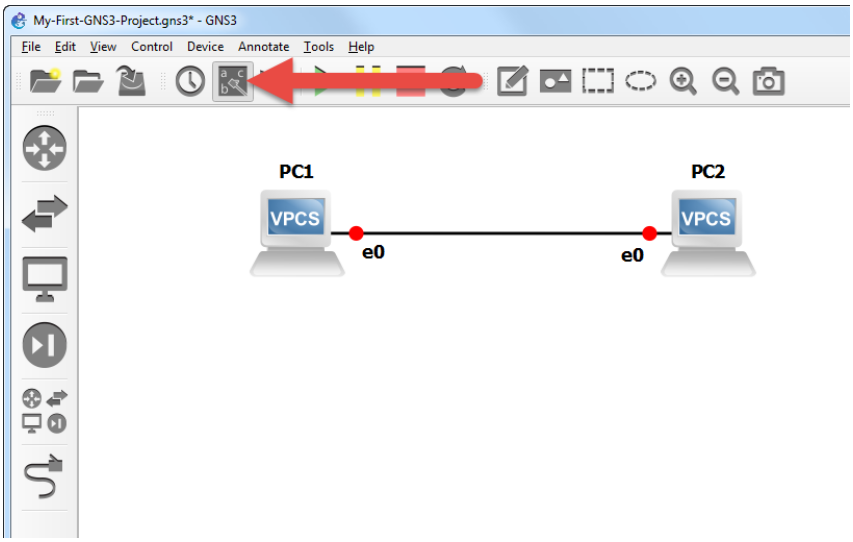


Figure 5.24: **Show/Hide Interface Labels** Button

8. Power on your network devices by clicking the **Start/Resume** button on the **GNS3 Toolbar** to start up your network devices:

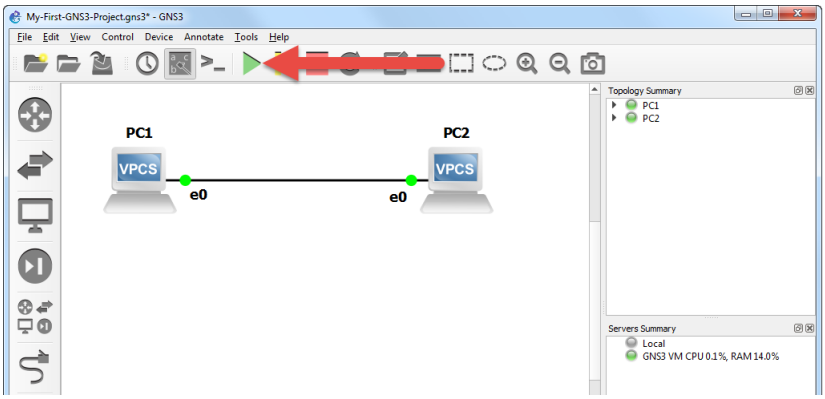


Figure 5.25: **Start/Resume** button

9. GNS3 indicates that the devices have been powered on by turning the interface connectors from red to green. This can also be seen in the **Topology Summary**:

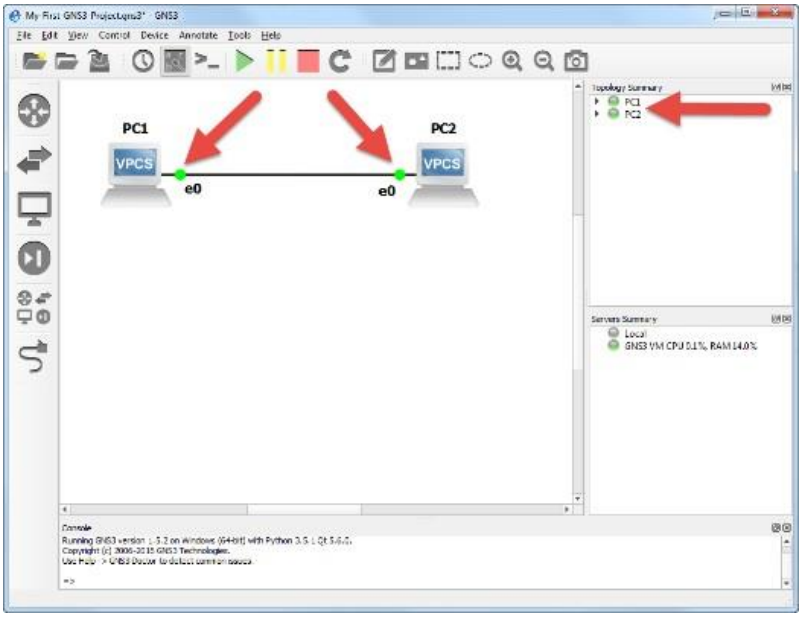


Figure 5.26: **Topology Summary**

10. You are now ready to configure your devices. Click the **Console connect to all devices** button on the **GNS3 Toolbar** to open a connection to every device in the topology:

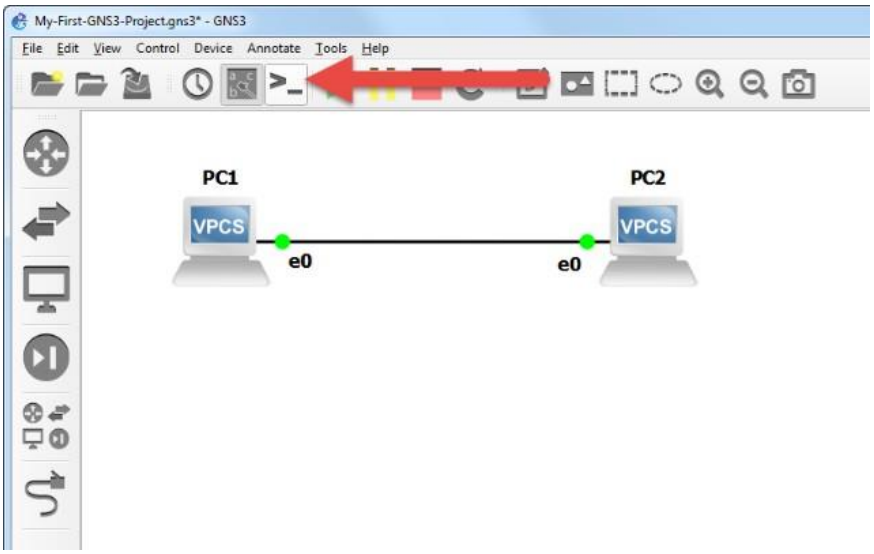


Figure 5.27: **Console connect to all devices** button

11.A console connection is opened to every device in the topology

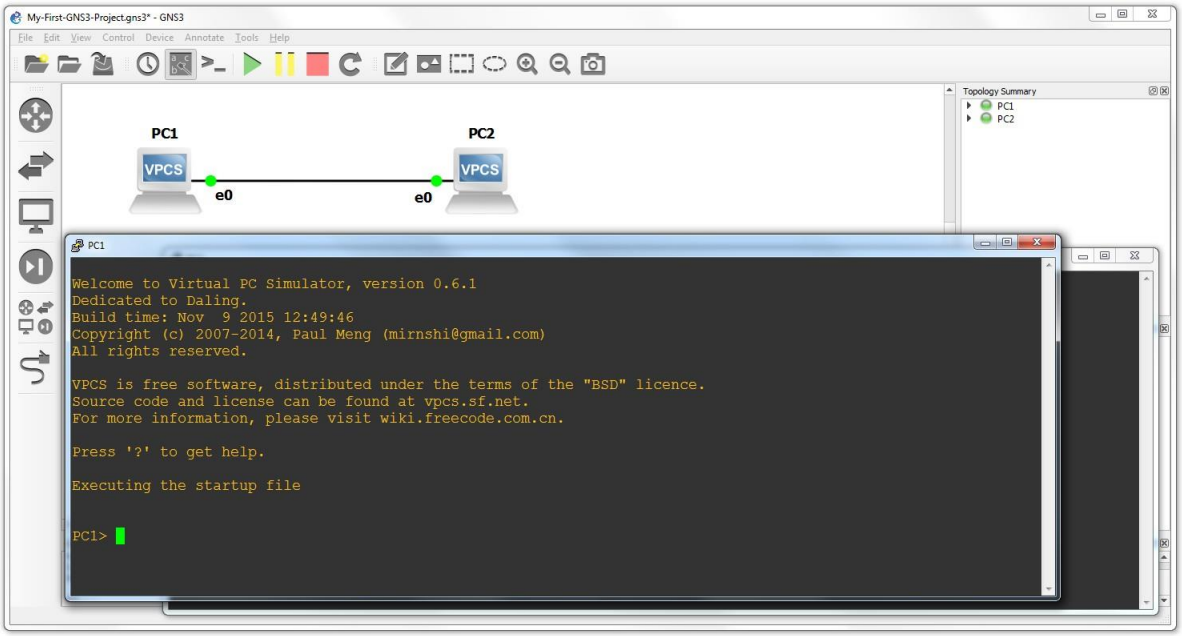


Figure 5.28: **Console connection**

12. Configure your PCs with IP addresses and default gateways as follows (a default gateway is configured in this example but is not used):

Syntax of IP address configuration:

ip ip-address network-mask default-gateway

PC1> ip 10.1.1.1 255.255.255.0 10.1.1.254

PC2> ip 10.1.1.2 255.255.255.0 10.1.1.254

13. **PC1** should now be able to ping **PC2** (use the key sequence **Ctrl-C** to stop the ping):

PC1> ping 10.1.1.2

Result Pings succeed.

14. To capture packets using Wireshark, right-click on the link and select **Start Wireshark.**

VI. LAB EXERCISES:

1. Design network configuration shown in Figure 5.29 for all parts. Connect all four VMs to a single Ethernet segment via a single hub as shown in Figure 5.29. Configure the IP addresses for the PCs as shown in Table 6.1.

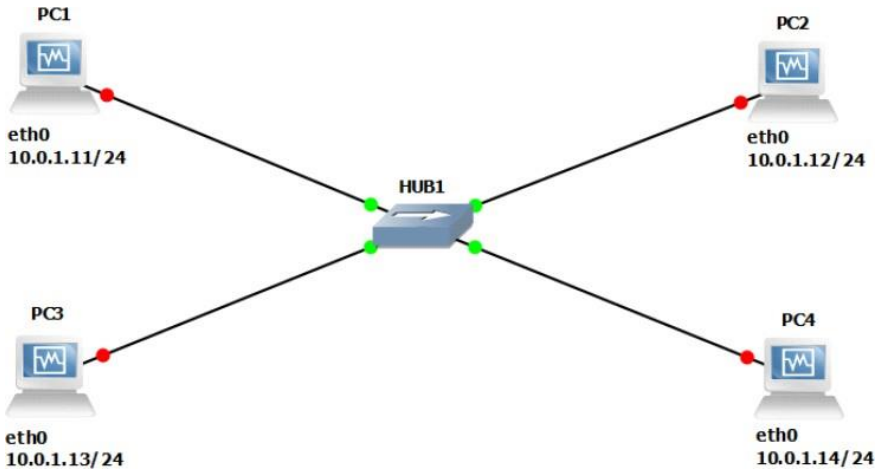


Figure 5.29: Network Design

VMS	IP Addresses of Ethernet Interface eth0
PC1	10.0.1.11 / 24
PC2	10.0.1.12 / 24
PC3	10.0.1.13 / 24
PC4	10.0.1.14 / 24

Table 5.1: IP Address of PCs

a. On PC1, view the ARP cache with show arp

b. Start Wireshark on PC1-Hub1 link with a capture filter set to the IP address of PC2.

c. Issue a ping command from PC1 to PC2: **PC1% ping 10.0.1.13 –c 3**

Observe the ARP packets in the Wireshark window. Explore the MAC addresses in the

Ethernet headers of the captured packets.

Direct our attention to the following fields:

• The destination MAC address of the ARP Request packets.

• The Type Field in the Ethernet headers of ARP packets.

d. View the ARP cache again with the command **arp -a**. Note that ARP cache entries can get refreshed/deleted fairly quickly (~2 minutes).

show arp

e. Save the results of Wireshark.

2. To observe the effects of having more than one host with the same (duplicate) IP address in a network.

After completing Exercise 1, the IP addresses of the Ethernet interfaces on the four PCs are as shown in Table 6.2 below. Note that PC1 and PC4 are assigned the same IP address.

VMS	IP Address of eth0
PC1	10.0.1.11 / 24
PC2	10.0.1.12 / 24
PC3	10.0.1.13 / 24
PC4	10.0.1.11 / 24

Table 5.2: IP addresses

a. Delete all entries in the ARP cache on all PCs.

b. Run Wireshark on PC3-Hub1 link and capture the network traffic to and from the duplicate IP address 10.0.1.11.

c. From PC3, issue a ping command to the duplicate IP address, 10.0.1.11, by typing

PC3% ping 10.0.1.11 –c 5

d. Stop Wireshark, save all ARP packets and screenshot the ARP cache of PC3 using

the arp –a command:

PC3% arp – a

e. When you are done with the exercise, reset the IP address of PC4 to its original value as given in Table 6.1.

3. To test the effects of changing the netmask of a network configuration.

a. Design the configuration as Exercise 1 and replace the hub with a switch, two hosts (PC2 and PC4) have been assigned different network prefixes.

Setup the interfaces of the hosts as follows:

VPCS	IP Address of eth0	Network Mask
PC1	10.0.1.100 / 24	255.255.255.0
PC2	10.0.1.101 / 28	255.255.255.240
PC3	10.0.1.120 / 24	255.255.255.0
PC4	10.0.1.121 / 28	255.255.255.240

b. Run Wireshark on PC1-Hub1 link and capture the packets for the following scenarios

i. From PC1 ping PC3.

ii. From PC1 ping PC2.

iii. From PC1 ping PC4.

iv. From PC4 ping PC1.

v. From PC2 ping PC4.

vi. From PC2 ping PC3.

c. Save the Wireshark output to a text file (using the “Packet Summary” option from “Print”) , and save the output of the ping commands. Note that not all of the above scenarios are successful. Save all the output including any error messages.

d. When you are done with the exercise, reset the interfaces to their original values as given

Table 6.1. (Note that /24 corresponds to network mask 255.255.255.0. and /28 to network

mask 255.255.255.240).

VII. EXERCISES

Based On Lab Question 1

• What is the destination MAC address of an ARP Request packet?

• What are the different Type Field values in the Ethernet headers that you observed?

• Use the captured data to analyze the process in which ARP acquires the MAC address for IP address 10.0.1.12.

Based On Lab Question 2

• Explain how the ping packets were issued by the hosts with duplicate addresses.

• Did the ping command result in error messages?

• How can duplicate IP addresses be used to compromise the data security?

• Give an example. Use the ARP cache and the captured packets to support your explanation.

Based On Lab Question 3

• Use your output data and ping results to explain what happened in each of the ping commands.

• Which ping operations were successful and which were unsuccessful? Why?

LAB No 6 **Date:**

Computer Network Design using SWITCH and ROUTERS in GNS3

Objectives:

- To Learn about IP address Assignment for different subnetworks
- To study the functions of ROUTER device
- To study the functions of SWITCH device

I. Introduction to Router Configuration

Setting Up of IOS Router:

Adding IOS Images to GNS3

Before you start creating projects using IOS routers, add at least one IOS image to GNS3 for example in Figure 6.1 , c3745 router image has been selected.

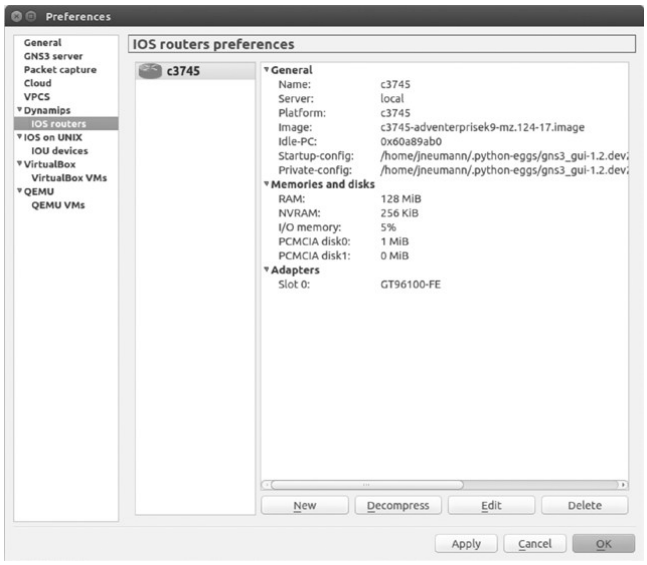


Figure 6.1: IOS routers preferences

Click **New** to start the wizard and then click the **Browse** button to locate your image file. After selecting your image file, you'll be asked whether you would like to decompress the IOS image

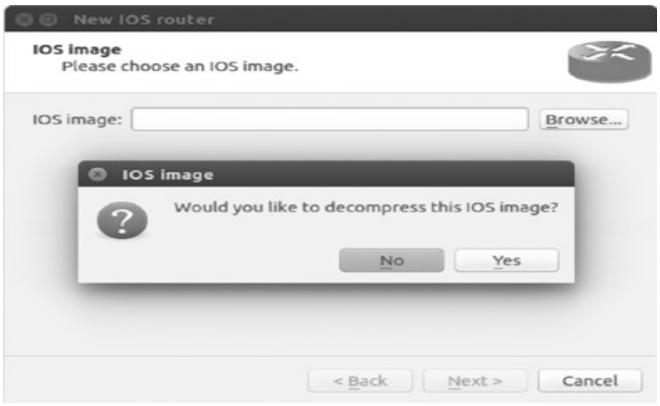


Figure 6.2: Deciding whether to decompress the IOS image

It's a good idea to let GNS3 decompress your image files; otherwise, your routers will have to decompress the images every time a router loads. Decompressing the images ahead of time will make your routers boot much faster. After decompressing your image, click **Next**, and GNS3 will attempt to recognize the router platform that your IOS belongs to, as shown in Figure 6.3.

Figure 6.3: Name and platform screen

GNS3 has determined that my image file belongs to a c3745 router platform and has automatically named it *c3745*.

In general, from here, you can just click through all the configuration settings to configure a basic router model, but the wizard provides opportunities for you to customize router memory and other features during this process. For now, click **Next** to continue. You should be presented with the Memory screen, shown in Figure 6.4.

Figure 6.4: IOS Memory screen

Your routers should run fine with the default memory setting. When you're done, click **Next**, and you will be presented with the Network adapters screen, as shown in Figure 6.5.

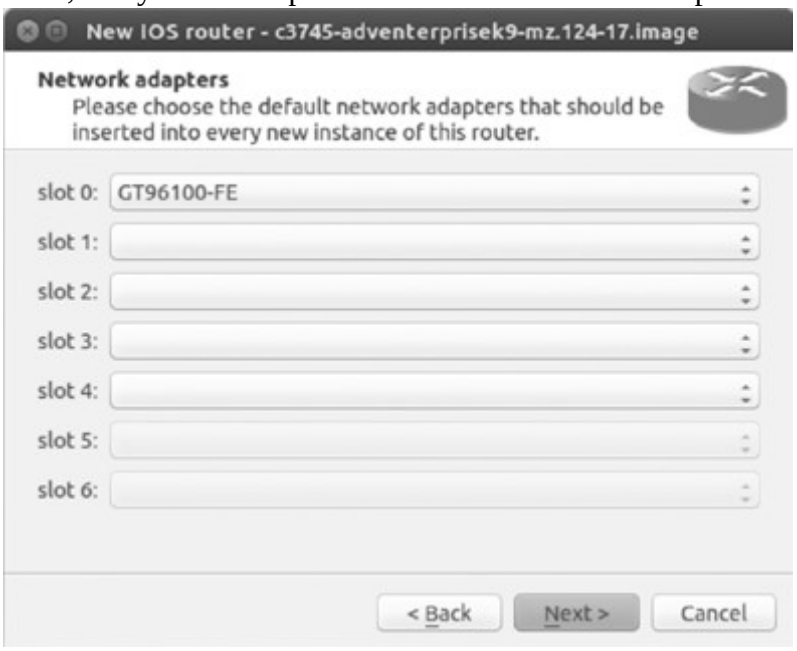


Figure 6.5: Network adapters screen

The default setting configures your router with the same standard options that are provided with a real model of the same Cisco router. If you would like to add more interfaces, use the drop-down menu next to the available slots and choose the desired network modules. The slot options will be limited to actual options that are available in the real version of the Cisco

router. When you're done, click **Next** and choose any WIC modules that you would like to install. Then click **Next** again to display the Idle-PC screen, shown in Figure 6.6.

Figure 6.6: Idle-PC screen

If you start a router in GNS3 without an Idle-PC setting, your computer's CPU usage will quickly spike to 100 percent and remain there. This happens because Dynamips doesn't yet know whether your virtual router is doing something that requires system resources, so it overcompensates by giving it all the resources it can. GNS3 will run sluggishly until this is corrected, and if CPU usage is left at 100 percent for a long time, your PC's processor could overheat.

You can easily fix this by having GNS3 look for places in the IOS program code where an idle loop exists (idle loops cause the CPU to spike); the result of this calculation is called an *Idle-PC value*. When the proper Idle-PC value is applied, Dynamips should periodically *sleep* the router when these idle loops are executed, which greatly reduces CPU usage.

To have GNS3 automatically find a value, click the **Idle-PC finder** button, and GNS3 will attempt to search for a value. If GNS3 finds a suitable value, then you're done; click **Finish**. If it's unsuccessful, leave the field blank and click **Next** to save the router without an Idle-PC configuration.

To Start with the Lab exercise, create a topology as shown in Figure 6.7:

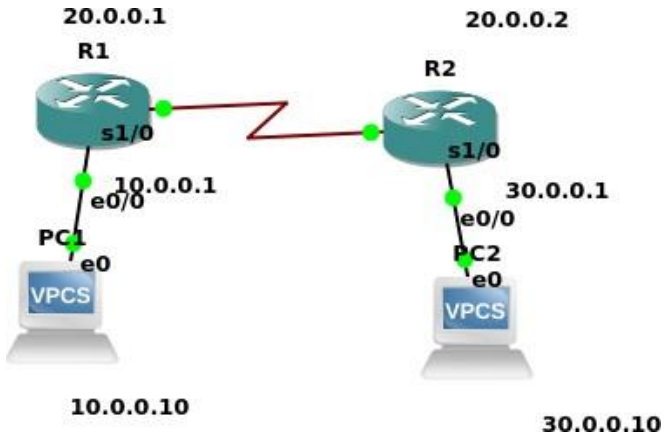


Figure 6.7: Network Topology

you will get a red light first.

II. LAB EXERCISES

1. Switching Cisco IOS Command Modes

This exercise demonstrates how to log into a router and how to work with the different Cisco IOS command modes. It is important to understand the different modes so you know where you are and what commands are accepted at any time.

i. Connect the Ethernet interfaces of the Linux PCs and the Cisco router as shown in Figure 6.7. Do not turn on the Linux PCs yet.

ii. Right-click on Router1 and choose Start.

iii. Right-click on Router1 and choose Console. Wait a few seconds until the router is initialized. If everything is fine, you should see the prompt shown below. This is the User EXEC mode. If the prompt does not appear, try to restart GNS3 and repeat the setup again.

Router1>

iv. To see which commands are available in this mode, type ?:

Router1>?

v. To view and change system parameters of a Cisco router, you must enter the Privileged EXEC mode by typing:

Router1>enable
Router1#

vi. Type the following command to disable the Privileged EXEC mode

Router1# disable

NOTE: The Cisco routers in GNS3 sometimes start up in Privileged instead of the User EXEC mode.

vii. To modify system wide configuration parameters, you must enter the global configuration mode. This mode is entered by typing:

Router1#configure terminal

Router1(config)#

or

Router1#conf t

Router1(config)#

viii. To make changes to a network interface, enter the interface configuration mode, with the command:

Router1(config)#interface FastEthernet0/0

Router1(config-if)#

The name of the interface is provided as an argument. Here, the network interface that is configured is FastEthernet0/0.

ix. To return from the interface configuration to the global configuration mode, or from the global configuration mode to the Privileged EXEC mode, use the exit command:

Router1(config-if)#exit

Router1(config)#exit

Router1#

The exit command takes you one step up in the command hierarchy. To directly return to the Privileged EXEC mode from any configuration mode, use the end command:

Router1(config-if)#end

Router1#

x. To terminate the console session from the User EXEC mode, type logout or exit:

Router1>logout

Router con0 is now available

Press RETURN to get started

2. Configuring a Cisco Router via the console

The following exercises use basic commands from the Cisco IOS that are needed to configure a Cisco router.

i. Right-click on Router1 and choose Start.

ii. Right-click on Router1 and choose Console. Wait some seconds until the initial console window is set up. When the router is ready to receive commands, proceed to the next step.

iii. Configure Router1 and Router 2 with the IP addresses given in Figure 6.7.

Note: In IOS Mode under Global Configuration,we can enable or disable IP Forwarding. When it is disabled it also deletes the contents of the routing table.

Router1(config)#ip routing
Router1(config)#no ip routing

In IOS Mode under Interface Configuration,we can enable or disable a network interface

Router1(config-if)#no shutdown
Router1(config-if)#shutdown

Tip: "no ip routing" is used to guarantee that the routing cache is empty, not routing table.

In Router 1

Interface Fastethernet0/0 in global configuration mode
R1(config)#inter f 0/0
R1(config-if)#ip address 10.0.0.1 255.0.0.0
R1(config-if)#no shutdown
R1(config-if)#exit

Interface Serial 2/0
R1(config)#inter s2/0
R1(config-if)#ip address 20.0.0.1 255.0.0.0
R1(config-if)#clock rate 64000
R1(config-if)#encapsulation ppp
R1(config-if)#no shutdown
R1(config-if)#exit
In Router 2

Interface Fastethernet 0/0
R2(config)#inter f0/0
R2(config-if)#ip address 30.0.0.1 255.0.0.0
R2(config-if)#no shutdown
R2(config-if)#exit

Interface Serial 2/0
R2(config)#inter s2/0
R2(config-if)#ip address 20.0.0.2 255.0.0.0
R2(config-if)#encapsulation ppp
R2(config-if)#no shutdown

R2(config-if)#exit

> **Tip to save Time:** It will be tiring to manually type in the configuration data for a router, everytime you set a lab, you can save time by saving all this configurations in an excel file and simply copying and pasting in the router console window.

iv. When you are done, use the following command to check the changes you made to the router configuration, and save the outputs:

R1# show interfaces

R1#show running-config

v. Assign ip addresses for both PC's as mentioned in Figure 6.7 with appropriate ip and subnet maskand default gateway.

3. Setting static routing table entries on a Cisco router

In this exercise, you will add static routes to the routing table of Router1. The routing table must be configured so that it conforms to the network topology shown in Figure 6.7. The routes are configured manually, which is also referred to as static routing.

The IOS command to configure static routing is ip route. The command can be used to show, clear, add, or delete entries in the routing table. The commands are summarized in the list below.

IOS MODE: PRIVILEGED EXEC
show ip route
Displays the contents of the routing table.
clear ip route *
Deletes all routing table entries
show ip cache
Displays the routing cache.
IOS MODE:GLOBAL CONFIGURATION
ip route cache / no ip route cache
Enables or disables route caching .By default , the route cache is enabled by the router.
ip route destination mask gw_address
no ip route destination mask
Adds or deletes a static routing table entry to the destination with netmask mask. The argument gw_address is the ip address of the next hop router.

> Note: Whenever an IP address is configured for a network interface on a router, routing table entries for the directly connected network are added automatically.

By default, Routers know only directed connected networks here Router 1 know only 10.0.0.0 and 20.0.0.0 it doesn't know the 30.0.0.0 like this R2 doesn't know about 10.0.0.0.So we are going to add Static route to this both router.

R1(config)#ip route Destination Network| Destination N/W Subnet Mask |Next Hop Address

In Router R1, just give this command, in this case Destination is 30.0.0.0 and its subnet mask is 255.0.0.0 next hop address is 20.0.0.2

R1(config)#ip route 30.0.0.0 255.0.0.0 20.0.0.2

In Router R2

R2(config)#ip route 10.0.0.0 255.0.0.0 20.0.0.1

Now both routers know all the networks.

i. Issue a ping command from PC1 to PC2, Router1 and PC4, respectively

ii. Save the captured Wireshark output.

iii. Use the saved data to answer the following questions:

• What is the output on PC1 when the ping commands are issued?

• Which packets, if any, are captured by Wireshark?

• Do you observe any ARP packets? If so, what do they indicate?

II. In the CSE department, two students sitting in two different labs want to establish a connection and send the data. So, configure the below network topology as shown in Figure. 6.8 and check the connectivity by pinging from PC0 to PC2.

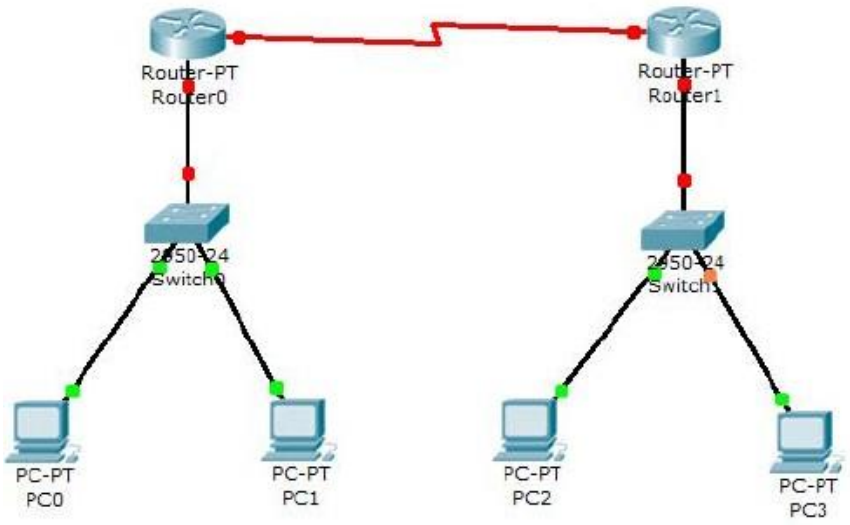


Fig: 6.8 Additional Lab Topology

LAB No 7 **Date:**

Study of Domain Name Server

Objectives:

- To illustrate the significance of Domain Name Server
- To Study the information exchanged between DNS and Clients

I. Introduction

DNS (Domain Name Servers)

Computers and other network devices on the Internet use an IP address to route the client request to required website. It's impossible for us to remember all the IP addresses of the servers we access every day. Hence we assign a domain name for every server and use a protocol called DNS to turn a user-friendly domain name like "howstuffworks.com" into an Internet Protocol (IP) address like 70.42.251.42 that computers use to identify each other on the network. In other words, DNS is used to map a host name in the application layer to an IP address in the network layer. DNS is a client/server application in which a domain name server, also called a DNS server or name server, manages a massive database that maps domain names to IP addresses. Client requests for address resolution which is defined as mapping a name to an address or an address to a name. It can be done in a recursive fashion or in an iterative fashion.

II. LAB EXERCISES

1. Configure the below topology to setup DNS server. R1 will use R2 as DNS server to make DNS resolutions.

First, lets begin with R1. We'll setup hostname and IP related information.

Figure 7.1 : Network Topology for DNS Configuration

R1 IP configurations:

Enable
configure terminal
hostname R1
interface e0/0
ip address 10.10.10.1 255.255.255.0
no shut
do wr
end

R2 IP and Hostname Configurations:

enable
config t
hostname R2
int e0/0
ip address 10.10.10.2 255.255.255.0
no shut
do wr
end
Setting up R2 as DNS Server

config t
ip dns server
ip host loopback.R2.com 2.2.2.2

We mapped loopback.R2.com to ip address 2.2.2.2. Currently, we don't have 2.2.2.2, we could create loopback interface on R2 and assign ip 2.2.2.2.
interface loopback 1

ip address 2.2.2.2 255.255.255.255

end

Let's verify that loopback interface we just created is working. This will show us that the hostname correctly setup locally on R2.

ping loopback.R2.com

Now it's time to setup R1 to resolve hostnames using R2. On R1 type;

config terminal

ip domain lookup
ip name-server 10.10.10.2

Set R1 to use R2 as default gateway to get to loopback interface on R2. So that after R1 resolves ***loopback.R2.com***, it can reach 2.2.2.2 through its default route (R2).
on R1 type:

config t
ip route 0.0.0.0 0.0.0.0 10.10.10.2
end

This tells our router that to get to any network not in it's routing table, it's next hop is 10.10.10.2 which is our router R2.

Now on R1, do a ping to ***loopback.R2.com*** and you should get a success message.

ping loopback.R2.com repeat 3

If you captured the traffic, you'll see DNS query and Answer as shown in Wireshark capture screen shot below.

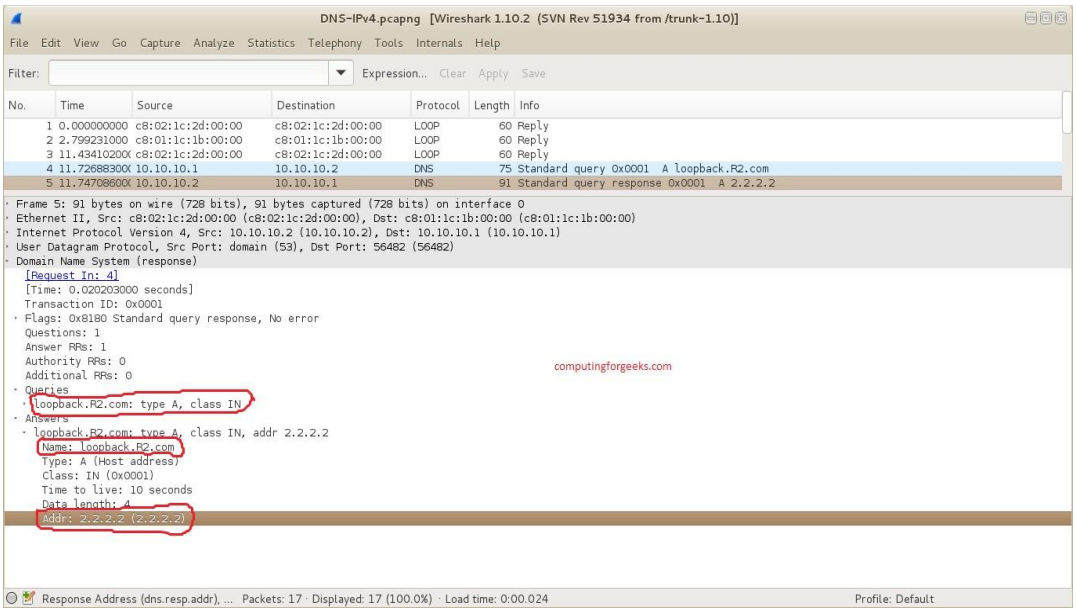


Figure 7.2 : Observation in WIRESHARK

III. LAB EXERCISE

Suppose you are connecting to www.mycsemit.com to read a page, you are a user sitting at a client's machine. You can access the www.mycsemit.com web server. The server machine finds the page you requested and sends it to you. Build a scenario using GNS3 to demonstrate the interaction of the DNS Server and DNS Client. Place DNS Server behind two routers.

LAB No 8 **Date:**

STUDY OF DHCP PROTOCOL

ServerObjectives:

- Understand DHCP Service
- Analyzing DHCP Packets
- Understanding significance of Netmask value

DHCP Overview

The Dynamic Host Configuration Protocol (DHCP) is based on the Bootstrap Protocol (BOOTP), which provides the framework for passing configuration information to hosts on a TCP/IP network. DHCP adds the capability to automatically allocate reusable network addresses and configuration options to Internet hosts. DHCP consists of two components: a protocol for delivering host-specific configuration parameters from a DHCP server to a host and a mechanism for allocating network addresses to hosts. DHCP is built on a client/server model, where designated DHCP server hosts allocate network addresses and deliver configuration parameters to dynamically configured hosts.

LAB EXERCISES

Configure two VMs that will be used to test connectivity from end to end and R1 will serve as a DHCP server to distribute IP addresses. The diagram below details the current setup:

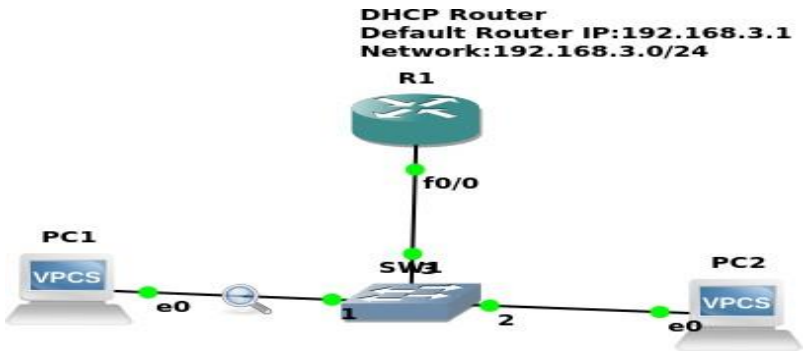


Figure 8.1 :Network Topology for DHCP Configuration

1. In order to configure our router as a DHCP server the following commands were used.

R1(config)#IP dhcp pool NAME

R1(dhcp-config)#Network 192.168.3.0 255.255.255.0

R1(dhcp-config)#Default-router 192.168.3.1

The commands above create a DHCP pool, adds the network that we want to assign IP addresses from, and specifies the default gateway for this subnet.

Note: There are many other parameters that go into configuring a DHCP server but this will suffice for our test environment.

That should be it for the DHCP configuration.

2.The next thing that you want to do is configure the fastethernet 0/0 interface which will connect to our switch.

R1(config)#Interface fastEthernet 0/0

R1(config-if)#No shutdown

R1(config-if)#ip address 192.168.3.1 255.255.255.0

The commands above will turn the interface on and assign an IP address.

3. Turn on the VPCS. In PC1 and PC2 type **dhcp**

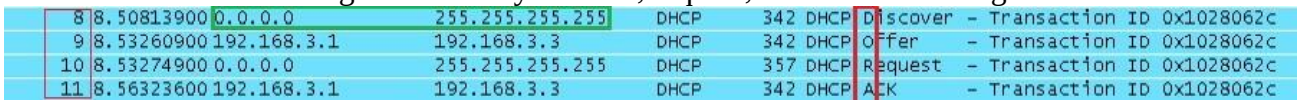
PC1>dhcp

PC2>dhcp

4. Let's analyze some of the traffic patterns using Wireshark.

In Wireshark we see the following information with regards to DHCP:

We see a discover message followed by an offer, request, and an acknowledgement. This is the process that clients go through in order to obtain an IP address via DHCP. The mnemonic for the steps above is DORA and it should help in memorizing the order of the steps.



8	8.50813900	0.0.0.0	255.255.255.255	DHCP	342	DHCP Discover - Transaction ID 0x1028062c
9	8.53260900	192.168.3.1	192.168.3.3	DHCP	342	DHCP Offer - Transaction ID 0x1028062c
10	8.53274900	0.0.0.0	255.255.255.255	DHCP	357	DHCP Request - Transaction ID 0x1028062c
11	8.56323600	192.168.3.1	192.168.3.3	DHCP	342	DHCP ACK - Transaction ID 0x1028062c

2. Network Prefixes and Routing

In this exercise, you study how the network prefixes (netmasks) play a role when hosts determineif a datagram can be directly delivered or if it must be sent to a router.

This part uses the network setup shown in Figure 8.2. The network includes one router, four hosts and two hubs. The IP addresses of all devices are given in Table 8.1. Here, each host hasonly a default route. In other words, the routing table at a host only knows about the directly connected

networks and the default gateway.

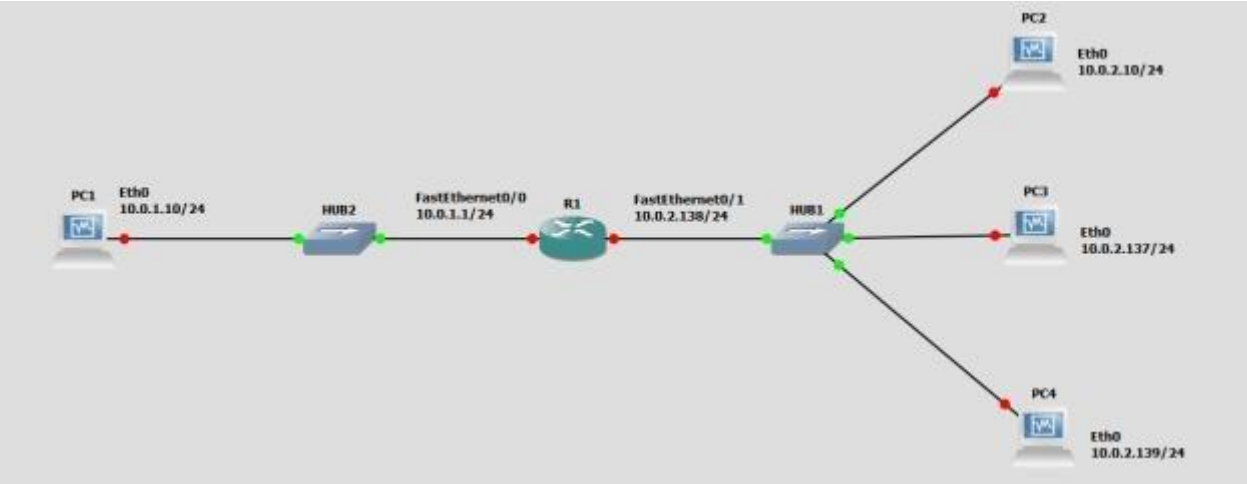


Figure 8.2: Network topology

Linux PC	Ethernet Interface eth0	Ethernet Interface eth1
PC1	10.0.1.10 / 24	Disabled
PC2	10.0.2.10 / 24	Disabled
PC3	10.0.2.137 / 29	Disabled
PC4	10.0.2.139 / 24	Disabled
Cisco Routers	FastEthernet0/0	FastEthernet0/1
Router1	10.0.1.1 / 24	10.0.2.138 / 24

Table 8.1

Exploring the role of prefixes at hosts

In this exercise, you explore how hosts that are connected to the same local area network, but that have different netmasks, communicate, or fail to communicate.

a. Configure the hosts and the router to conform to the topology shown in Figure 8.2, using the IP addresses as given in Table 9.1. Note that PC2, PC3, and PC4 have different netmasks.

b. Add Router1 as default gateway on all hosts. (PC1, PC2, PC3, and PC4).

c. Issue *ping* commands from PC1

a i) Clear the ARP table on all PCs.

a ii) Start Wireshark on PC1 and on PC3, and set the capture filter to capture ICMP and

b ARP packets only.

a iii) Issue a ping command from PC1 to PC3 for at least two sends (-c 2).

a iv) Save the output of the ping command at PC1 and the output of Wireshark on PC1 and PC3.

a v) Save the ARP tables, routing tables, and routing caches of each host. Please note that

b these are the tables entries from Step 3 after the ping commands are issued.

d. Issue *ping* commands from PC3 to PC4

a i) Clear the ARP table on all PCs.

a ii) Start Wireshark on PC3, and set the capture filter to capture ICMP and ARP packets

b only.

a iii) Check the ARP table, routing table, and routing cache of each host. Save the output.

b Please note that these are the table entries from Step 4 before the ping is issued.

a iv) Issue a ping command from PC3 to PC4 for at least three sends (-c 3) .

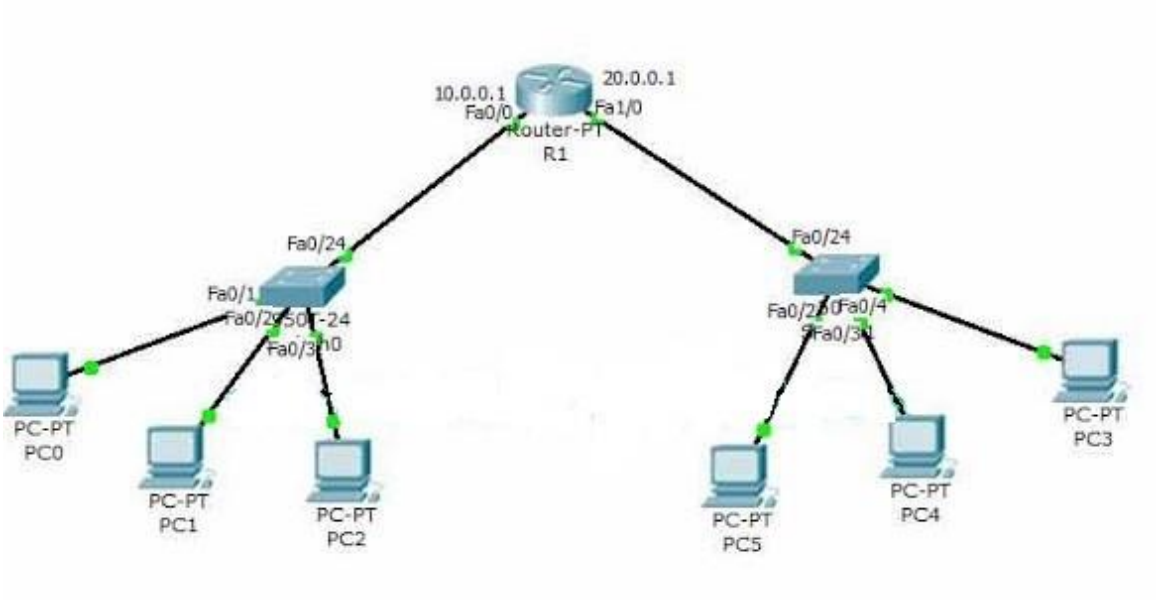
a v) Save the output of the ping command and the output of Wireshark on PC3.

a vi) Save the ARP table, routing table, and routing cache of PC3. Please note that these are

b the table entries from Step 4 after the ping commands are issued.

5. Repeat Step 4, but this time issues a ping from PC3 to PC2. Note that once an entry is made in the routing cache, you cannot repeat the previous experiment to obtain the same results. You have to wait until the routing cache is reset or you can delete all the routing caches on all devices.

II. In an institute, there is one DHCP server and two departments that want IP addresses for the end users. A DHCP client could request an IP address and DHCP server must respond to client requests as the server is always active. So, configure DHCP for the below configuration. Also, show the configuration if connecting one more DHCP server with the current DHCP server.



LAB-09 **Date:**

Design of VLANs Using GNS3

Objectives:

- To understand Virtual Lan (VLAN) Concepts

We can solve many of the problems associated with layer 2 switching with VLANs. VLANs work like this: Figure 11.1 shows all hosts in this very small company connected to one switch, meaning all hosts will receive all frames, which is the default behavior of all switches.

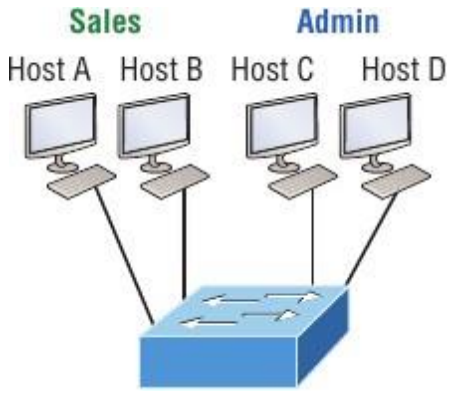


Fig 11.1 *One switch, one LAN: Before VLANs, there were no separations between hosts.*

If we want to separate the host's data, we could either buy another switch or create virtual LANs, as shown in Figure 11.2

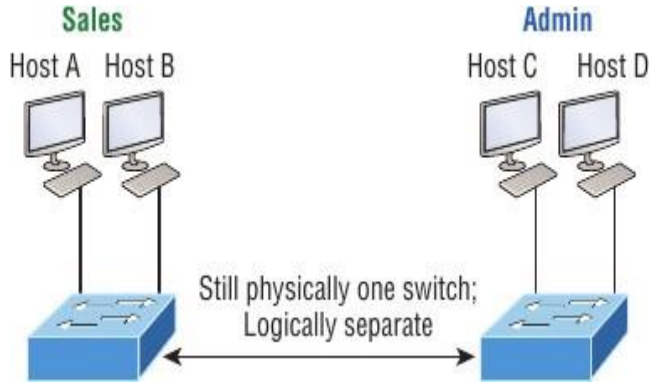


Fig 11.2 *One switch, two virtual LANs (logical separation between hosts): Still physically one switch, but this switch acts as many separate devices.*

In Figure 11.2 , we configured the switch to be two separate LANs, two subnets, two broadcast domains, two VLANs—they all mean the same thing—without buying another switch. We can do this 1,000 times on most Cisco switches, which saves thousands of Rupees and more!

There are two different types of ports in a switched environment. Let's take a look at the first type in Figure 11.3

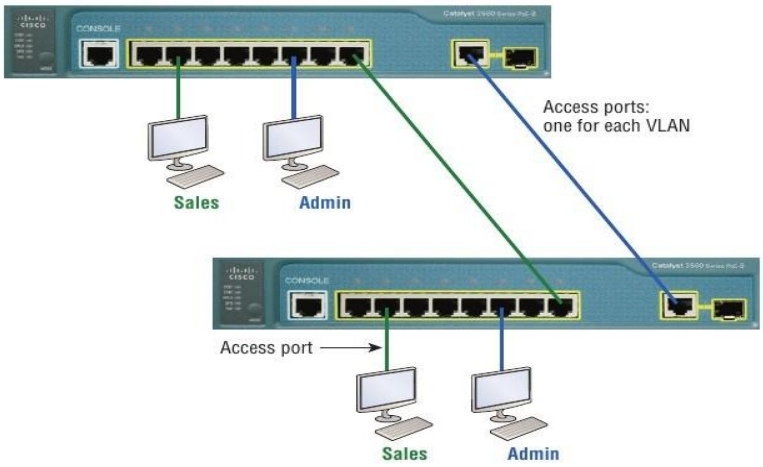


Fig 11.3 *Access Ports*

Notice there are access ports for each host and an access port between switches—one for each VLAN.

Access ports

An access port belongs to and carries the traffic of only one VLAN. Traffic is both received and sent in native formats with no VLAN information (tagging) whatsoever. Anything arriving on an access port is simply assumed to belong to the VLAN assigned to the port. Because an access port doesn't look at the source address, tagged traffic—a frame with added VLAN information—can be correctly forwarded and received only on trunk ports.

added VLAN information—can be correctly forwarded and received only on trunk ports.

With an access link, this can be referred to as the configured VLAN of the port. Any device attached to an access link is unaware of a VLAN membership—the device just assumes it's part of some broadcast domain. But it doesn't have the big picture, so it doesn't understand the physical network topology at all.

Another good bit of information to know is that switches remove any VLAN information from the frame before it's forwarded out to an access-link device. Remember that access-link devices can't communicate with devices outside their VLAN unless the packet is routed. Also, you can only create a switch port to be either an access port or a trunk port—not both. So you've got to choose one or the other and know that if you make it an access port, that port can be assigned to one VLAN only. In Figure 12.3, only the hosts in the Sales VLAN can talk to other hosts in the same VLAN. This is the same with Admin VLAN, and they can both communicate to hosts on the other switch because of an access link for each VLAN configured between switches.

Trunk ports

The term trunk port was inspired by the telephone system trunks, which carry multiple telephone conversations at a time. So it follows that trunk ports can similarly carry multiple VLANs at a time as well.

A trunk link is a 100, 1,000, or 10,000 Mbps point-to-point link between two switches, between a switch and router, or even between a switch and server, and it carries the traffic of multiple VLANs—from 1 to 4,094 VLANs at a time. But the amount is really only up to 1,001 unless you're going with something called extended VLANs.

Instead of an access link for each VLAN between switches, we'll create a trunk link demonstrated in Figure 11.4. Trunking can be a real advantage because with it, you get to make a single port part of a whole bunch of different VLANs at the same time. This is a great feature because you can set ports up to have a server in two separate broadcast domains simultaneously, so your users won't have to cross a layer 3 device (router) to log in and access it.

Another benefit to trunking comes into play when you're connecting switches. Trunk links can carry the frames of various VLANs across them, but by default, if the links between your switches aren't trunked, only information from the configured access VLAN will be switched across that link.

It's also good to know that all VLANs send information on a trunked link unless you clear each VLAN by hand.

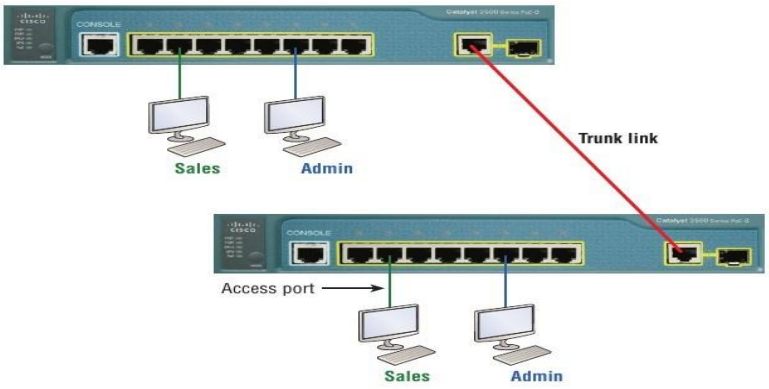


Fig 11.4 *VLANs can span across multiple switches by using trunk links, which carry traffic formultiple VLANs.*

Frame Tagging

As you now know, you can set up your VLANs to span more than one connected switch. You can see that going on in Figure 11.4, which depicts hosts from two VLANs spread across two switches. This flexible, power-packed capability is probably the main advantage to implementing VLANs, and we can do this with up to a thousand VLANs and thousands upon thousands of hosts!

All this can get kind of complicated—even for a switch—so there needs to be a way for each one to keep track of all the users and frames as they travel the switch fabric. And this just happens to be where frame tagging enters the scene.

This frame identification method uniquely assigns a user defined VLAN ID to each frame.

Here's how it works: Once within the switch fabric, each switch that the frame reaches must first identify the VLAN ID from the frame tag. It then finds out what to do with the frame by looking at the information in what's known as the filter table. If the frame reaches a switch that has another trunked link, the frame will be forwarded out of the trunk-link port.

Once the frame reaches an exit that's determined by the forward/filter table to be an access link matching the frame's VLAN ID, the switch will remove the VLAN identifier. This is so the destination device can receive the frames without being required to understand their VLAN identification information.

Another great thing about trunk ports is that they'll support tagged and untagged traffic.

simultaneously if you're using 802.1q trunking. The trunk port is assigned a default port VLAN ID (PVID) for a VLAN upon which all untagged traffic will travel. This VLAN is also called the native VLAN and is always VLAN 1 by default, but it can be changed to any VLAN number. Similarly, any untagged or tagged traffic with a NULL (unassigned) VLAN ID is assumed to belong to the VLAN with the port default PVID. Again, this would be VLAN 1 by default. A packet with a VLAN ID equal to the outgoing port native VLAN is sent untagged and can communicate to only hosts or devices in that same VLAN. All other VLAN traffic has to be sent with a VLAN tag to communicate within a particular VLAN that corresponds with that tag.

VLAN Identification Methods:

1. Inter-Switch Link (ISL)

Inter-Switch Link (ISL) is a way of explicitly tagging VLAN information onto an Ethernet.

frame. This tagging information allows VLANs to be multiplexed over a trunk link through an external encapsulation method. This allows the switch to identify the VLAN membership of a frame received over the trunked link.

2. IEEE 802.1q

Created by the IEEE as a standard method of frame tagging, IEEE 802.1q actually inserts a field into the frame to identify the VLAN. If you're trunking between a Cisco switched link and a different brand of switch, you've got to use 802.1q for the trunk to work.

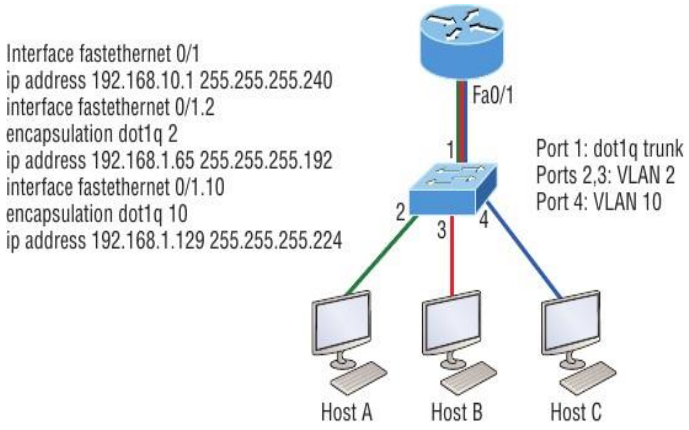
Unlike ISL, which encapsulates the frame with control information, 802.1q inserts an

802.1q field along with tag control information.

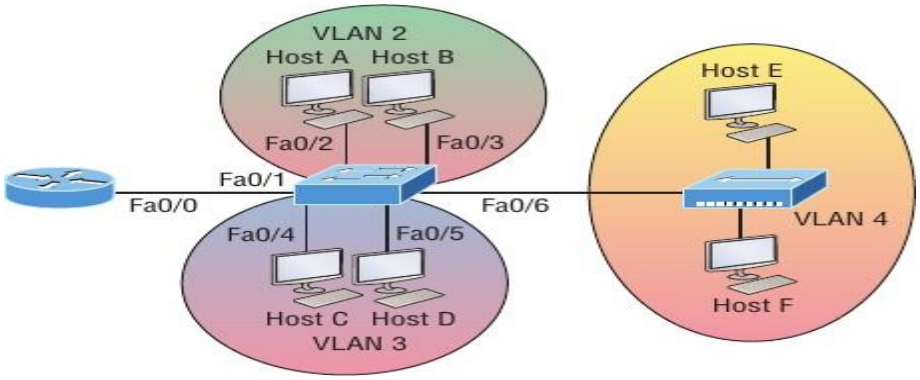
LAB EXERCISE

Configure following inter-VLAN example in GNS3 and verify the working using Wireshark tool.

1.



2.



LAB No 10 **Date:**

Study of Dynamic Routing Protocols using GNS3

Objectives:

- To study the routing information protocol.
- To study the open shortest path first.

1. Routing Information Protocol - RIP:

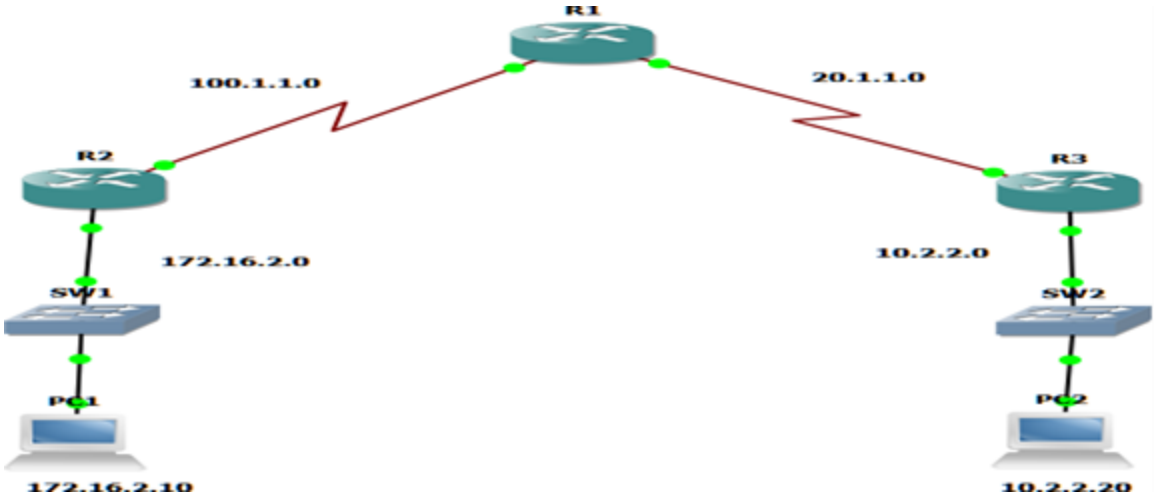
Routing Information Protocol (RIP) is a dynamic routing protocol which uses hop count as a routing metric to find the best path between the source and the destination network. It is a distance vector routing protocol.

Hop Count: Hop count is the number of routers occurring in between the source and destination network. The path with the lowest hop count is considered as the best route to reach a network and therefore placed in the routing table. RIP prevents routing loops by limiting the number of hopes allowed in a path from source and destination. The maximum hop count allowed for RIP is 15 and hop count of 16 is considered as network unreachable.

Features of RIP:
1. Updates of the network are exchanged periodically.
2. Updates (routing information) are always broadcast.
3. Full routing tables are sent in updates.
4. Routers always trust routing information received from neighbor routers. This is also known as Routing on rumors.

RIP versions:
There are three versions of routing information protocol – RIP Version1, RIP Version2 and RIPng. **RIP v1** is known as *Classful* Routing Protocol because it doesn't send information of subnet mask in its routing update.
RIP v2 is known as *Classless* Routing Protocol because it sends information of subnet mask in its routing update.
RIPng (RIP next generation) is an extension of RIPv2 for support of IPv6, the next generation Internet Protocol.

The RIPv2 routing protocol uses the following command syntax:



Router(config)#router rip

Router(config-router)#version 2

Router(config-router)#network <network-IP>

Router(config-router)#network <network-IP>

LAB EXERCISE:

Configure the below topology to setup connectivity using RIPv2. R1, R2, and R3 will use dynamic routing protocol (RIPv2).

Configuration for R1

R1#conf t

R1(config)#int s1/0

R1(config-if)#ip add 100.1.1.2 255.255.255.0
R1(config-if)#no shut

R1(config-if)#int s1/1

R1(config-if)#ip address 20.1.1.1 255.255.255.0

R1(config-if)#no shut

R1(config-if)#exit

R1(config)#router rip

R1(config-router)#version 2

R1(config-router)#network 20.1.1.0

R1(config-router)#network 100.1.1.0

Configuration for R2

R2#config t

R2(config)#int f1/0

R2(config-if)#ip address 172.16.2.1 255.255.0.0

R2(config-if)#no shut

R2(config-if)#int s2/0

R2(config-if)#ip address 100.1.1.1 255.255.255.0

R2(config-if)#no shut

R2(config-if)#exit

R2(config)#router rip

R2(config-router)#version 2

R2(config-router)#network 172.16.0.0

R2(config-router)#network 100.1.1.0

Configuration for R3

R3#config t

R3(config)#int s2/0

R3(config-if)#ip add 20.1.1.2 255.255.255.0

R3(config-if)#no shut

R3(config-if)#int f1/0

R3(config-if)#ip add 10.2.2.1 255.255.255.0

R3(config-if)#no shut

R3(config-if)#exit

R3(config)#router rip

R3(config-router)#ver 2

R3(config-router)#network 10.2.2.0

R3(config-router)#network 20.1.1.0

RIP Verification:

show ip route command should display all RIP networks and end to end ping should be successful.
show ip protocol command should display if necessary, ports are active.
show ip rip database command should displays the contents of RIP database inside the router.
debug ip rip command shows RIP updates occurring in the system undebug all Once you turn on *debug ip rip* router will keep showing RIP updates. The command undebug all will stop such RIP updates.
show running-config command is used to get the current configuration from the Router.

2. Open Shortest Path First - OSPF:

BASIC OSPF - Enable OSPF

Open Shortest Path First (OSPF) is an IGP developed by the OSPF working group of the Internet Engineering Task Force (IETF). Designed expressly for IP networks, OSPF supports IP subnettingand tagging of externally derived routing information. OSPF also allows packet authentication and uses IP multicast when sending/receiving packets.

As with other routing protocols, enabling OSPF requires that you create an OSPF routing process, specify the range of IP addresses to be associated with the routing process, and assign area IDs to be associated with that range of IP addresses.

Configuration

With OSPF, every router has its own unique "picture" (topology map) of the network. Routers use "HELLO" packets to periodically check with routers to ensure they are still there. Every router in OSPF is identified with a "router ID". The router ID can be manually entered or OSPF will automatically choose the IP address with the highest number. It supports variable length

subnet masks (VLSM), making it a classless routing protocol.

OSPF works well in point to point and point to multipoint, broadcast or non-broadcast configurations. OSPF also offers a number of OSPF-specific features such as stub areas, virtual links, and OSPF on demand circuits. In OSPF route redistribution is supported between different routing protocols.

An OSPF point-to-multipoint interface is defined as a numbered point-to-point interface having one or more neighbors. It creates multiple host routes. An OSPF point-to-multipoint network has the following benefits compared to no broadcast multiaccess and point-to-point networks.

The OSPF routing protocol uses the following command syntax:

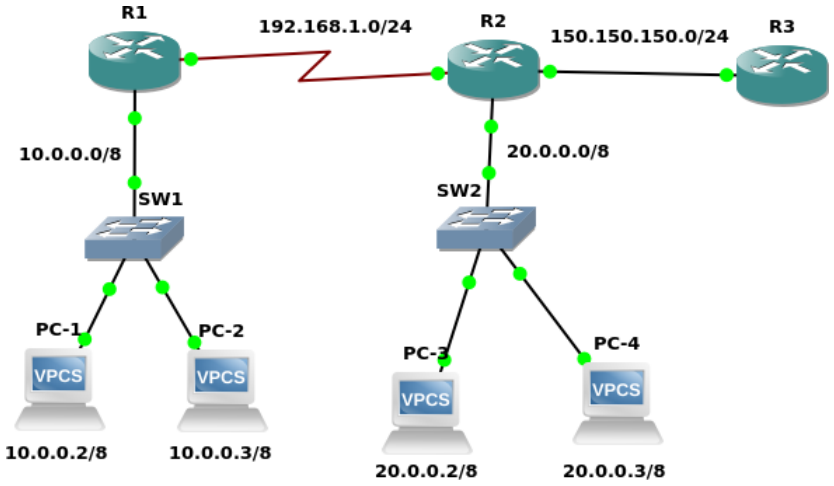
Router(config)#router ospf <process id 1-65535>

Router(config-router)network <network address> <wild card mask> area <0-4294967295>

Area id number can always be zero (0) for small networks, but for larger networks, the area IDs need to be properly planned as all routing updates must traverse area 0.

LAB EXERCISE:

Configure the below topology to setup connectivity using RIPv2. R1, R2, and R3 will use dynamic routing protocol (OSPF).



Configuration for R1

```
R1(config)#router ospf 200
R1(config-router)#network 10.0.0.0 0.255.255.255 area 0
R1(config-router)#network 192.168.1.0 0.0.0.255 area 0.0.0.0
R1(config-router)#exit
```

Configuration for R2

```
R2(config)#router ospf 200
R2(config-router)#network 20.0.0.0 0.255.255.255 area 0
R2(config-router)#network 192.168.1.0 0.0.0.255 area 0
R2(config-router)#network 150.150.150.0 0.0.0.255 area 1
R2(config-router)#exit
R2(config)#exit
```

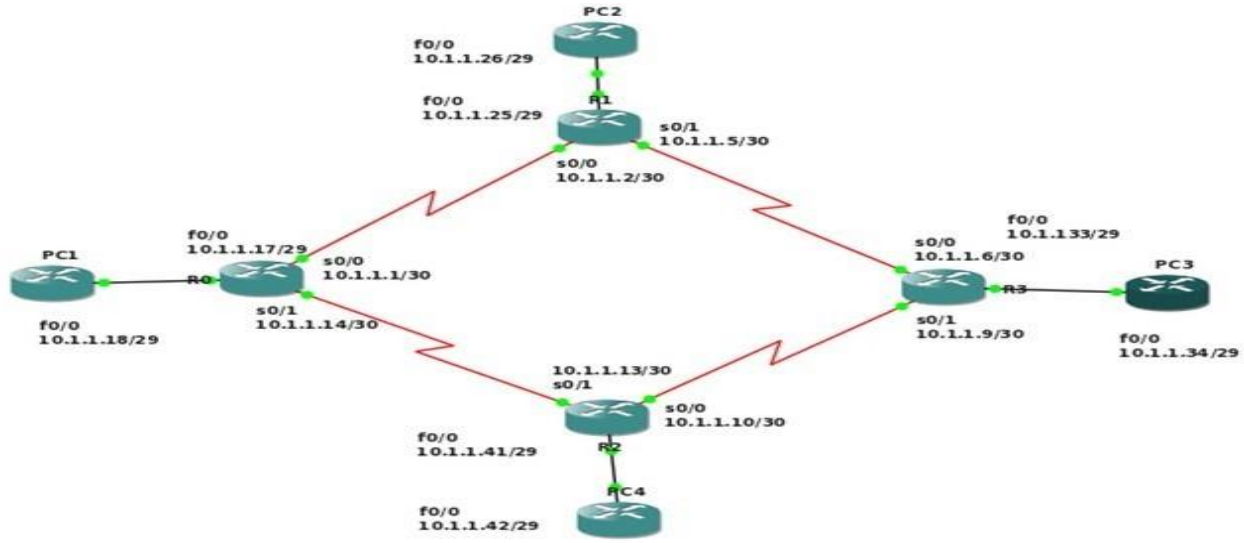
Configuration for R3

```
R3(config)#router ospf 200
R3(config-router)#network 150.150.150.0 0.0.0.255 area 1
R3(config-router)#exit
R3(config)#exit
```

OSPF Verification:

```
show ip route
show ip ospf neighbor
show ip ospf database
```

II. Configure the below network topology using RIP and OSPF as shown in Figure and check the connectivity by pinging from PC1 to PC2, PC3, PC4.



****

LAB No 11 **Date:**

MINI PROJECT DEMONSTRATION AND EVALUATION

LAB No 12 **Date:**

END SEMESTER LAB EXAMINATION

References:

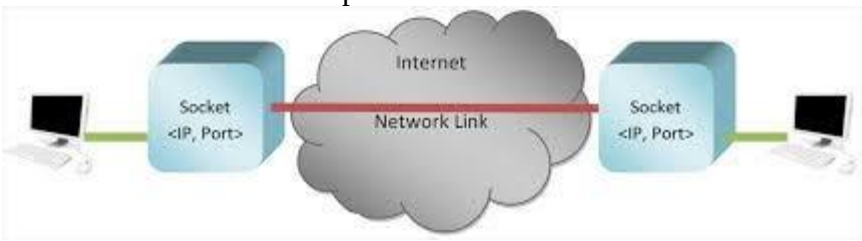
1. W. Richard Stevens," UNIX Network Programming, Volume 1: The Sockets Networking API", Third Edition,Addison-Wesley Professional Computing, 2003.
2. Introduction and Reference Guide to Wireshark, https://thepracticalsysadmin.com/wireshark-reference-guide.
3. Jason C. Nuemann, "The Book of GN3", No Starch Press,2015.
4. GNS3 Documentation, https://www.gns3.com.

APPENDIX-I

Mini Project Topics

1. Since lot of progress has been made in the field of video technology, video communication is gradually changes from point-to-point to multipoint-to-multipoint or from normal image quality to high-definition image quality. Design a project based on the basic concept of socket programming which is used to establish a connection between client and server during a video call.
a) The video must be streamed from one-client device to another server device and vice-versa.
b) The main goal of the project is to use the socket structure of a UDP datagram to connect the host and client for an efficient video-calling system.

Description about project:
Here you have to create a client-server model where server create sockets on startup and clients are connected with servers. A client should know the server IP & port.



For Video Data Transmission project:
At Server Side:
a) Acquire video frames of webcam using OpenCV or Python.
b) With pickle (process where a object hierarchy is converted into a byte stream) serialize frame to byte data.
c) Pack (used to fill the entire frame) each frame data using certain module(struct).
d) Send data to client and display frame
At Client Side:
a) Receive frames and append them to data
b) Unpack the data using struct module
c) Load the frame using pickle
d) Display the frame at Client side

2. Implement the project on network statistics such as Throughput, Average RTT, Transmission Speed. So, measure the throughput and RTT for a TCP client and server program. Also, find the RTT through ping command.
Description about project:
In this project, a tcp client server program must be designed where first a connection is established from client to server. The server then sends a segment (in bytes) to the server. The sending time and RTT are calculated based on sending information.
On server side, two structure variables (using struct) t1 and t2 must be declared. Calculate the time elapsed for both variables i.e. t1 and t2 in seconds.
On client side, structure variables (using struct) t1 and t2 calculates the receiving time of data coming from the server.
With the help of client side and server side implementation, calculate the throughput and RTT time.

3. Write and implement the project on application that read packet that travels across various layers of

Transmission Control Protocol/Internet Protocol (TCP/IP) model of network architecture. The packet sniffer will analyse the network traffic that allows users to get a practical understanding of the flow of packets in a network. You can use various Application layer protocols such as HTTP, DNS, Transport Layer: TCP, UDP and Network Layer: IPv4.

Description about project:

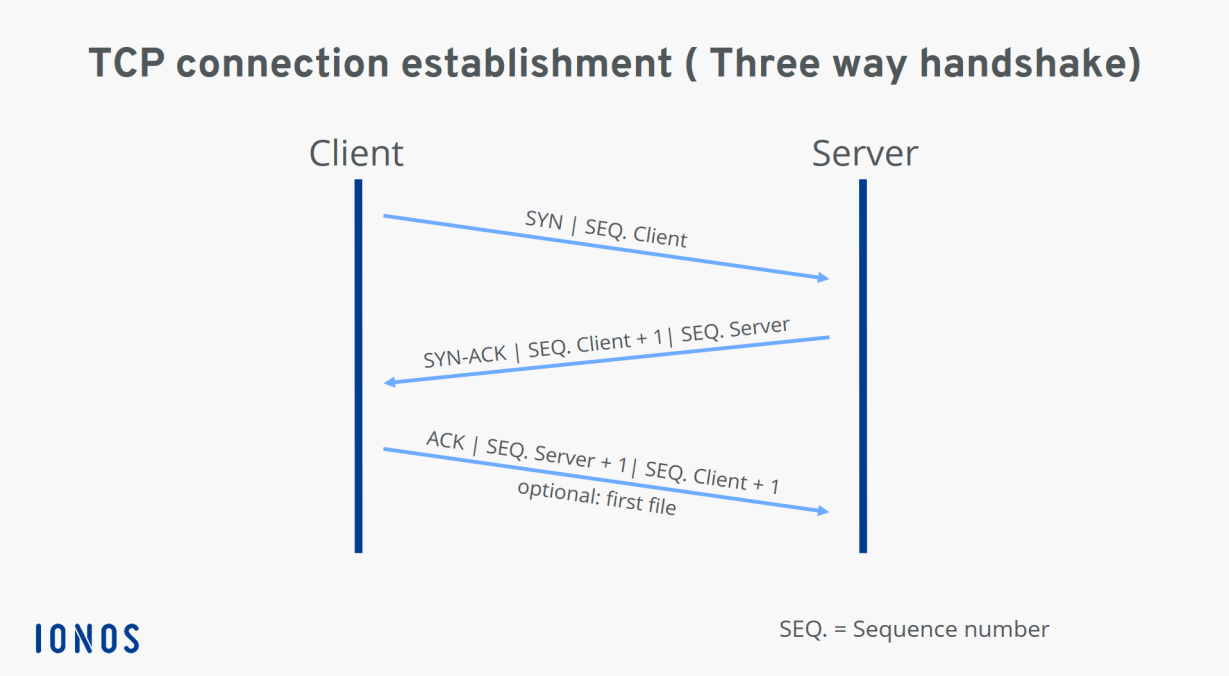
- In first step, you have to create function for opening of socket and listen for the packets in the process.
- Information about the packets is passed on to a function that processes the ethernet protocol.
- Strip the header based on data fields, passes it to higher-level protocol (IPv4, ARP etc.). After that processed TCP, UDP protocols.
- Parse DNS, HTTP, SMTP protocols for further packet transmission.
- Analysing the results using packet sniffer and read/check the illegal access and read encrypted/Unencrypted data.

4. Design a Tool for capturing the traffic based on the TCP header flags (ACK, SYN, FIN, RST, PSH, URG). Use Wireshark for simulation purpose to capture and analyse packets based on the TCP header flags.

Description about project:

The actual process for establishing a connection with the TCP protocol is as follows:

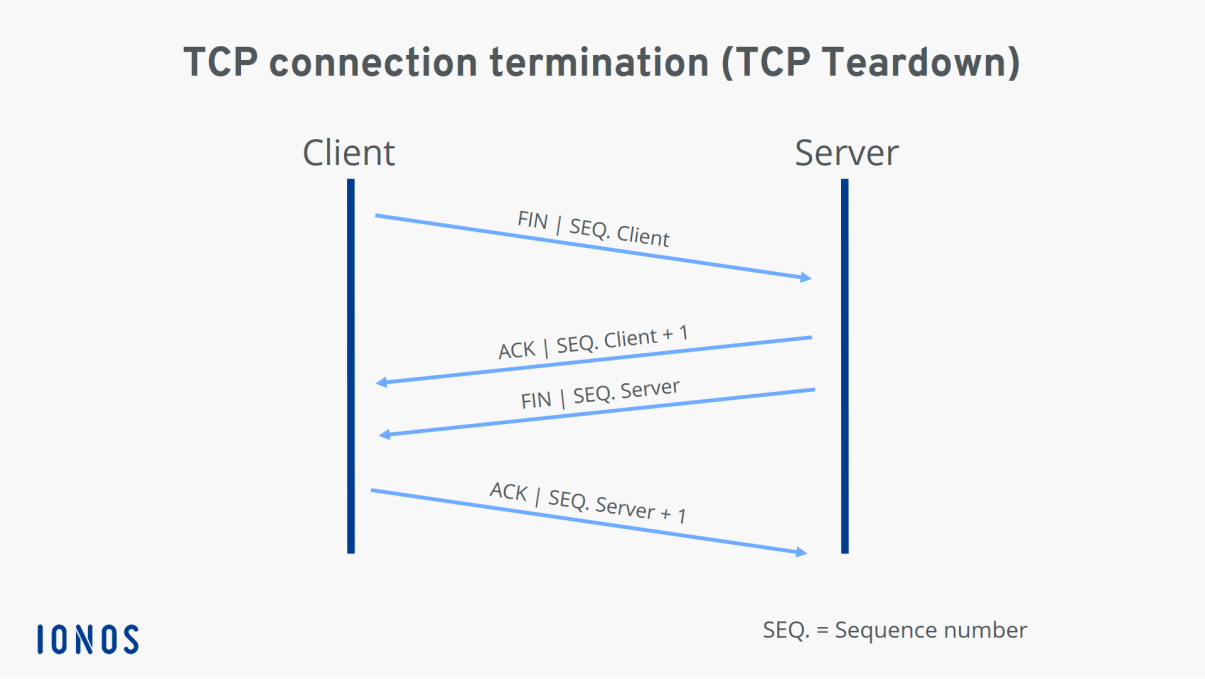
1. First, the requesting client sends the server a SYN packet or segment (SYN stands for synchronize) with a unique, random number. This number ensures full transmission in the correct order (without duplicates).
2. If the server has received the segment, it agrees to the connection by returning a SYN-ACK packet (ACK stands for acknowledgment) including the client's sequence number plus 1. It also transmits its own sequence number to the client.
3. Finally, the client acknowledges the receipt of the SYN-ACK segment by sending its own ACK packet, which in this case contains the server's sequence number plus 1. At the same time, the client can already begin transferring data to the server.



Both sides of a connection can terminate a TCP connection, and even one-sided termination is also possible. This is also known as a half-open connection, whereby the other side is still allowed to transfer data even if one side has already disconnected.

The individual steps of two-way termination (initiated by the client for the sake of simplicity in this example) can be summarized as follows:

1. The client sends a FIN segment to notify the server that it no longer wants to send data. It sends its own sequence number, just as it does when the connection is established.
2. The server acknowledges receipt of the package with an ACK segment that contains the sequence number plus 1.
3. When the server has finished the data transfer, it also sends a FIN packet, to which it adds its sequence number.
4. Now it is the client's turn to send an ACK packet including the sequence number plus 1, which officially terminates the TCP connection for the server.


TCP connection termination (TCP Teardown)
Client
Server
FIN | SEQ. Client
ACK | SEQ. Client + 1
FIN | SEQ. Server
ACK | SEQ. Server + 1
IONOS
SEQ. = Sequence number

APPENDIX-II

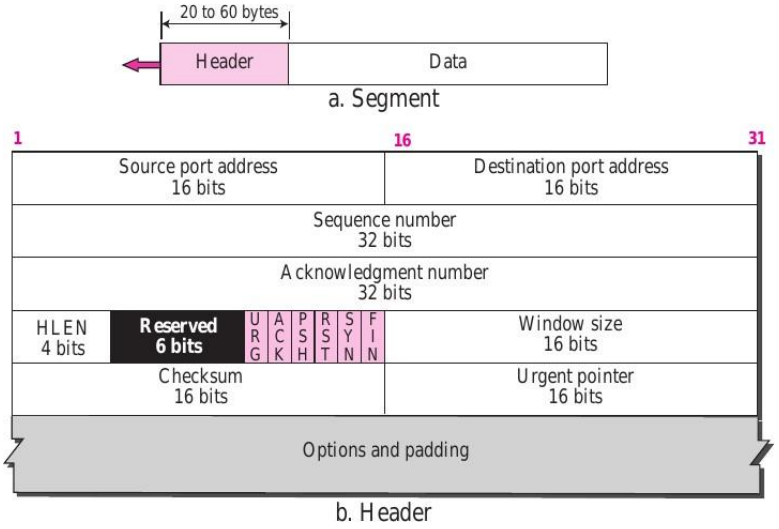


Fig 1: TCP Segment Format

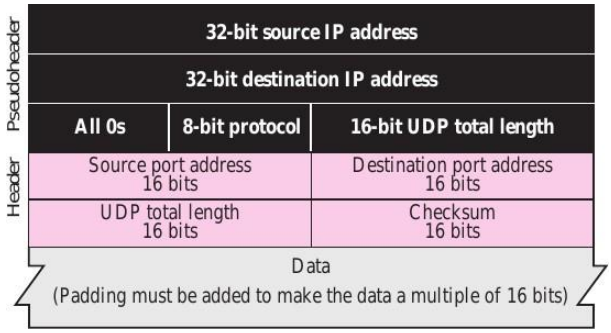


Fig 2: UDP Header and Pseudoheader

Fig 3: Format of DNS message

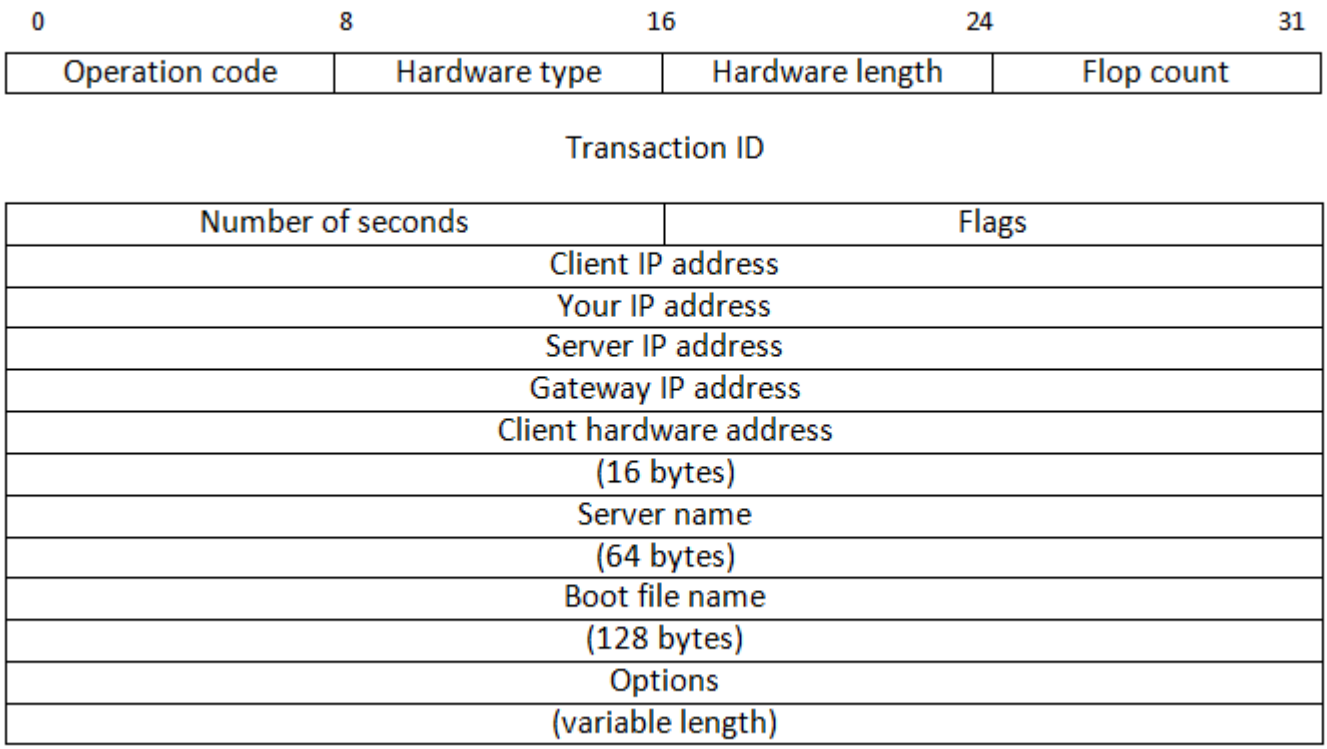


Fig 4: Format of DHCP Packet

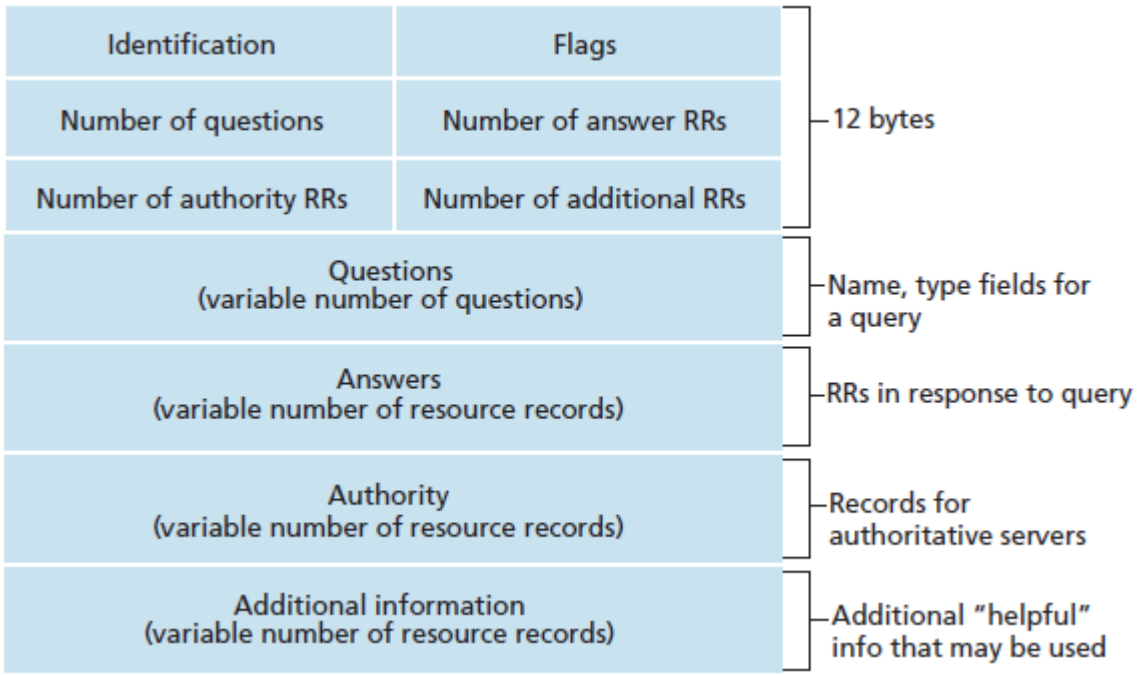


Fig 5: DNS Message Format